COMPUTERS
FOR
BEGINNERS

COMPUTERS FOR BEGINNERS

R. THIAGARAJAN
Joint Adviser
Department of Science & Technology
Government of India
New Delhi 110016



STERLING PUBLISHERS PVT. LTD.

STERLING PUBLISHERS PRIVATE LIMITED
L-10, Green Park Extension, New Delhi-110016
G-2, Cunningham Apartments, Cunningham Road, Bangalore-560052
Mishradeep, Patliputra Path, Rajendra Nagar, Patna-800016

Computers for Beginners

First edition, 1984
Second Revised & Enlarged edition 1986
Reprint, 1988
Reprint, 1991

PRINTED IN INDIA

Published by S.K. Ghai, Managing Director, Sterling Publishers Pvt. Ltd., L-10, Green Park Extension, New Delhi-110016. Printed at Roopak Printers, Delhi-110032.

DEDICATED
TO
AMMA

Preface to First edition

Electronic computers have come to stay. Starting as scientific wonders occupying enormous space and radiating heat in the late forties, computers today are now found as Personal Computers(P.C.s), which can be owned by many of us and, carried in brief cases. Such a transformation is the result of concerted efforts of scientists, engineers and indeed, the community at large.

Though India did not lag far behind in introducing computers by following closely on the heels of the development elsewhere in the world, progress of computerisation in the country has not been satisfactory at all. Efforts are now being made to introduce computers in different activities what with the growth in the number of indigenous manufacturers of late.

Computers have to be understood by all sections of society so that the full potentials of these machines can be exploited. When computer professionals try to popularise computers, they find that no good introductory book for beginners on computers is available in the market. This book is an attempt at explaining the various facets of electronic computers in simple terms.

The author was invited to deliver computer courses to senior government officials during the past few years. In addition, the author has taught school children in a few schools of Delhi at a time (1978-84) when computers had not entered the country in a big way. The lecture material used in these courses has been modified and updated, wherever necessary, and is now in a reasonable shape for use even by a layman and indeed, a child going to secondary school.

With the plans of the Government of India for introducing computers in the schools, it is hoped that this book would become a "Computer Primer" textbook. With that hope in view, a number of questions have also been provided at the end of every chapter.

Yes, one may notice that this introductory book does not contain information on micro-processor based systems. One could have introduced a chapter on this topic but the author feels that the logic used by a human being while utilising computers remains more or less the same whatever Hardware one may use. Thus, problem solving has been given more prominence in preference to description of hardware.

This book would not have become a reality but for the polite persuasion of organisers of computer courses and the active participation by the distinguished officials and indeed the enthusiastic and indomitable school children who attended such courses which this author had the privilege to address.

This introductory book is, therefore, dedicated to the intellectuals

of all age groups whether they attended any of the author's courses or not.

The author would welcome comments and criticisms on the subject content of the book from readers.

New Delhi-110029 R.Thiagarajan

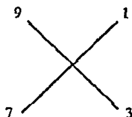
Preface to Second edition

It is heartening to note that the book has been found useful by the public in general and school teachers and school children in particular. Though many suggestions have been received with regaro to incorporation of examples relating to programming languages, one does not find much merit in such a move.

The book has been updated to some extent.

While looking at the historical growth of computing, one finds that our forefathers in India had developed a number of tricks! Let us see how they performed multiplication without the use of tables. An example will illustrate this point.

Whenever you wanted to multiply a single digited number greater than 5 by another similar number, our forefathers said there was no need to remember the multiplication tables! How then do you multiply 9 by 7? This is how you do it. Put 9 on one end of a diagonal of a cross and 7 at the other diagonal as shown in Figure. Then take the difference between 10 and 9 and put that at the end of the diagonal having 7 at its end.



Do a similar operation to 7.
(You are expected to know tables upto 5X5).

You now multiply 3 by 1 to give 3. Then subtract either 3 from 9 or 1 from 7 to get 6. Insert 6 in front of 3 to give 63. This is called the "sluggard method".

Another illustration will reinforce the versatility of the thinkers of a yester era in this land.

'It is said that in the holy city of Varanasi, Brahma called the pandits to his abode at the time of creation of the human species. Brahma then told them that he was giving three needles to them as shown in Figure. On one of the needles he placed 64 disks one on top of the other as shown.

CONTENTS

1

Introduction to Computers

1. Introduction

1.1 Since the dawn of civilisation, man has been manipulating data in one form or the other. In fine, data would represent facts, figures and recorded observations. Data was essentially used to extract meaningful information in order that decisions could be taken for action. Thus, information turned out to be data which was organised, sifted, collated and offered in time. Different people saw the same data in different light. With the advent of the industrial revolution (wherein man's muscular powers were supplemented by machines), the appetite for accurate information in different fields of human endeavour grew by leaps and bounds.

1.2 The birth of electronic computers to help decision-makers at all levels has truly been acclaimed as the second industrial evolution. Electronic computers heralded the advent of augmentation of man's mental abilities. While one often hears of a computer replacing the human brain, it may safely be assumed that the intricacies of the human brain are too complex to be replaced by a machine at present.

1.3 Computers are being used today for a host of applications ranging from such routine jobs as payroll preparation to sophisticated problems such as control of a lunar capsule. Computer usage can be broadly classified into three categories:

(a) scientific computation
(b) process control and
(c) business data processing.

2. History of Computers

2.1 Mechanical calculators could be called the fore-runners of modern digital computers. The *Abacus* is perhaps the oldest and the most important of these calculators. In the hands of a skilled operator, it can outshine some

of the modern desk calculators. The abacus looks somewhat like the one shown in Fig. 1.

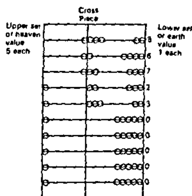


Fig 1. Abacus

The beads or counters represent digits Each upper counter is worth five units and each lower counter is worth one unit. The value of the digit in each position is determined by adding the values of the beads pressed against the crosspiece

2.2 In the mid-seventeenth century, Blaise Pascal, a well known mathematician and philosopher, used the position of a rotating wheel to indicate the value of a digit in a mechanical calculator he developed. Near the end of seventeenth century, Leibnitz, who was also a mathematician and philosopher, extended Pascal's ideas to design a desk calculator that could multiply and divide also.

2.3 In 1812, Charles Babbage, Professor of Mathematics at Cambridge University, devised a machine that could automatically calculate trigonometric and logarithmic tables. It was called a *Difference Engine* because it used a *difference method* of computation. Babbage received a large government grant to develop his difference engine, but the project turned out to be something of a flop because it was beyond the technological capabilities of the time.

2.4 Babbage also conceived of another calculator called the *Analytical Engine*. This calculator was to have two distinct parts --- a store to hold data until they were to act as operands and a mill to process the data. Answers were to be returned to the store for future reference. This in principle, is the way modern digital computers work. The analytical engine, like the difference engine, was never completed.

2.5 In the late nineteenth century, Herman Hollerith, an employee of the Census Bureau of U.S., realised that punched cards could be used to store digital information by the presence or absence of a series of holes on the card. Once information is stored in this form, machines could help in the enormous task of enumeration inherent in the taking of a census. Hollerith's scheme was adopted by commercial interests in the decades that

followed, and prior to the development of computers, it was widely applied to the processing of large volumes of data in business and statistical applications. Though Hollerith did not foresee the electronic computer, his punched card till today has been one of the most important means for entering digital information into a computer. Of course, other media such as floppies and cassettes have taken its place now.

2.6 In 1937, George Stibnitz of the Bell Telephone Laboratories built an electromechanical calculator, which used open or closed relays to record information rather than using the presence or absence of holes. The first relay computer called the *complex calculator*, was set up in the Bell Laboratories in 1940.

2.7 The ultimate in elctromechanical calculators was designed by Howard Aiken at Harvard University in 1937. This machine was called Automatic Sequence Controlled Calculator --- Mark I, which prepared mathematical tables by automatically performing a set sequence of arithmetic operations. The Mark I, completed in 1944, is historically important because it was the immediate predecessor of the electronic computer, and contained many features, such as the pre-established programme of operations now associated with computers

The Electronic Digital Computer

2.8 In 1946, a computer using electronic components instead of mechanical relays was developed by J. Presper Eckert and John W Mauchly of the Moore School of Engineering of the University of Pennsylvania. This was called the Electronic Numerical Integrator and Calculator (ENIAC). Though it was programmed by means of switches and plug-in connections, because of its electronic components, it is often identified as the first electronic computer. It was used mainly for preparing mathematical tables.

2.9 ENIAC was installed at the Aberdeen Proving Ground and helped to produce many tables, especially ballistic tables, for the Ordnance Department of the U.S. Army. The ENIAC used vacuum tubes to record data in lieu of mechanical relays or holes in cards or positions of wheels. It was programmed by means of boards and switches that established new connections for a myriad of cables on the floor. It did not keep both the instructions and the data in its storage, but only data. It was, however, much faster than the Mark I, and its success led to a computer boom in the period 1947 to 1952.

2.10 After the ENIAC, many research laboratories, most of them connected with universities, began to construct computers. One of the most active of these research groups was headed by Eckert and Mauchly at the Moore School of Engineering. They designed the EDVAC (Electronic Discrete Variable Automatic Computer) which differed fundamentally from

the ENIAC in two ways --- the use of binary numbers and the internal storage of instructions written in digital form.

2.11 In 1954, the first installation of an electronic digital computer for business applications was made at the General Electric Appliance Park in Louisville, Kentucky. This computer was called the Universal Automatic Computer or UNIVAC. This computer is important because it was the first commercially available computer. The initial opinion of many of the scientists who developed computers was that a very small number of computers would suffice for the entire U.S. However, business organisations soon recognised their potential data processing value. And thus, this became an important factor in the growth of the computer industry.

Computer Generations

2.12 Computers built during the period of the UNIVAC I to the late 1950s used vacuum tubes, and are identified as first generation computers. IBM, which had not been particularly active in the development of computers, entered the computer business with IBM 701 in the year 1953. Late in 1954, IBM installed the first of the IBM 650 computers. This small-to-medium-scale computer was the most popular one during the next five years. In this period, IBM attained a dominant position in the computer field, with more than two-thirds of the market. The first generation computers, in the main, gave rise to problems of accommodation on the one hand and difficulties in dissipating the heat generated by the vacuum tubes on the other.

2.13 The emergence of transistors as viable substitutes for vacuum tubes in the early sixties brought about the second generation of computers. As is common knowledge, a transistor can perform the same functions as a vacuum tube, is less expensive, smaller in size, and generates literally no heat. Thus, the second generation computers were smaller in size, faster, required less power and needed less air-conditioning. Some of the second generation computers are IBM-1401, ICL-1901 etc.

2.14 While transistor circuits were faster than vacuum tubes, their speed was not adequate enough. Hence, research was directed towards faster circuit components. As an offshoot of the development of miniaturized circuits, the prices of computer components also began to fall. Concurrent with the development of components and circuits, facilities for usage of one computer by more than one person simultaneously were also provided. Sophistication in data communication techniques and development of operating systems to keep a record of, and control of, computer processing also brought a step towards the third generation. IBM's system 360 etc., introduced in the late sixties, can be considered to be the third generation machines.

Table 1
Comparison of Computer Generations

Sl. No.	Level	I	II	III	IV	V	VI
1.	Year of introduction	1946	Early 50's	Early 60's	Early 70's	1982	1990
2.	Electronic circuit Components	Thermionic Valves	Transistors	ICs	Lsi	VLSI	Biological ?
3.	Number of users	1	1	Multiple user community	Remote users	Same as IV generation	Not Known
4.	Languages used	Machine Language (low-level)	Symbolic Language, FORTRAN	COBOL, ALGOL, PL/1 etc.	PASCAL, Query ,Word	Kamel Language, Query	Natural Language of the user ?
5.	Nature of inputs	Torn tapes	Punched cards, MT, Disks etc.	Cards, MT, Disks.	VDU's Floppies, Optical Disks, COM's	Optical Disks, Optical records	Speech Signs, Thought waves ?
6.	Nature of processing	Serial	Serial	Serial	Serial/ Parllel/ Array	Parallel/ Array	Associative ?
7.	Cooling arrangement	Extensive	Moderate	Less	Room A/C adequate	Room A/C	In-built Liquid Coolers ?
8.	Space requirements	Large	Medium	Small	Normal Table top PCs	Office room	Hand-held ?

2.15 There have been improvements in the third-generation computers, but there is no dramatic breakthrough which could be called the "fourth generation." The seventies have brought a change towards the growth of small computers called *Minis*. These computers are being manufactured by companies which are very small compared to computer giants like IBM. These small companies have introduced innovative design techniques in the machines and are promoting usage by even very small business enterprises.
2.16 The "Fifth Generation" is said to be still in an experimental stage having been initiated in 1979. It is hoped that the "Fifth Generation" would facilitate Natural Language Processing and recognition of visual symbols.

The Indian Setting

2.17 In our own country, a public sector undertaking called Electronics Corporation of India Limited (ECIL), located at Hyderabad started manufacturing computers in the late sixties. Today, ECIL is marketing a number of computers ranging from those for data processing to the ones meant for process control. These machines are called TDC machines and a number of them are being used in our country for various applications today. A short list of computer groups providing micro-processor based systems (and minis) is given at the end of the book.

Due to a number of steps taken by the Department of Electronics, Government of India, in the early seventies, a number of system engineering groups in India have been putting together microprocessor-based systems.

3. Types of Computers

Definition of Computer

3.1 The term "Computer" can be logically applied to any calculating device. However, there are some distinguishing characteristics as given below:

(a) *Electronic*: The computer uses electronic elements — transistors, resistors diodes etc. It operates with two state logic based on two measurable states in these electronic components.

(b) *Internal storage*: The computer has an internal storage (memory) to store both the programme and the data being processed by the programme.

(c) *Stored programme*: The programme of instructions which spcifies the sequence of operations to be followed is stored in

the internal memory. It is the stored programme which makes the computer "automatic"because the entire set of steps to be taken is determined in advance, and no human invention is required during execution.

(d) *Programme modification* : A distinguishing feature of the computers is the ability to change the stored programme of instructions during the execution of programme steps.The modification is usually based on the form, quanity, or value of the data being processed.

In summary, the computer is an electronic computational device having an internal storage, a stored programme of instructions, and the capability for modification of the set of instructions during the execution of the programme.

Classification of Computers

3.2 There are two types of computers, namely, the Analogue Computer and the Digital Computer. The analogue computer measures a quantity (just like a voltmeter). The computer acts on some inputs, performing a number of mathematical operations or solving an equation, and plots the output data on a graph. An analogue computer, like a slide rule, is simpler and more direct than a digital computer, but is not ordinarily a high precision device.

3.3 The thermometer, for instance, records various mercury levels based on changes in the temperature. When the temperature rises, the mercury level rises and when the temperature falls, so does the mercury level. That is, changes in the temperature are shown by analogues or like-changes in the level of mercury in the thermometer. The same principle of similarity is used by analogue computers in performing arithmetic functions with measurements. Numbers are represented by physical quantities. Measurements are made with results displayed and later used in various arithmetic operations to generate similar new data.

3.4 The analogue computer performs at its best when a precision of three or four digits is sufficient, programme flexibility is not important, and there is only one independent variable. Data available in analogue form, such as a varying voltage representing temperature or speed can be fed directly to an analogue computer. However, its disadvantages-- lack of precision and limited flexibility-- are usually not sufficient to overcome the advantage. To run a different problem, an analogue computer usually must be reconnected by means of interconnecting patch cords.

Digital Computers

3.5 A digital computer, on the other hand, counts, rather than measures.

It is really an ultrahigh-speed electronic calculating machine. One way of looking at the difference between the two is to say that a digital computer works with digits while the analogue one does not. The earliest known digital computer consists of the ten fingers of man which even today are being extensively used for counting. But the electronic devices give us extremely high speeds in counting. They suffer from one drawback, however, and that is, their inability to occupy anything other than two known stable states--*on* or *off*. Thus, digital computers, as we know them today, operate on binary digits as opposed to decimal digits with which we all are familiar. Because numbers are used, we can hope to get any degree of precision by increasing the number of digits. Again, because of the binary number system, *ON* can be made to represent a 1, while *OFF* could represent a 0.

3.6 Almost all computers encountered by us in routine applications are digital, and hence, we shall confine our attention to this type in these discussions.

Categorization According to Orientation

3.7 Digital computers could be classified further into the following categories:

(a) Business data processing.
(b) Scientific computation.
(c) Process control.

These distinctions do not always hold too clearly, because the trend is to make a general-purpose computer which can be used for any of these purposes A brief summary of these characteristics, however, is given below.

3.8 *Business data processing* typically requires the processing of large numbers of records but with relatively little computation for each record There is also a large volume of output in the form of reports. The computer that is oriented towards business data processing will, therefore, emphasize high-speed input and output and instructions to facilitate the conversion (editing) of stored data into report format.

3.9 *Scientific computation* is characterized by relatively little input or output but extensive internal computation Scientific computers will emphasize high-speed internal processing but may have limited input and output capabilities

3.10 *Process control* requires the computer to accept frequent inputs of data from the process being controlled such as a chemical process, to test the data and to make computations to adjust the process. The computer will usually be servicing several controls, so that the hardware is designed to accept and act on many concurrent or simultaneous inputs.

4. Digital Computer Components

4.1 Most digital computers can be represented by the block diagram as given in Fig.2. The basic units are:

(a) Arithmetic and logic section, which performs the actual arithmetic operations;

(b) Memory or Storage, which retains the programme (instructions), problem and solution;

(c) Control unit, which directs the computer's operation;

(d) Input devices for translating all input information into usable form; and

(e) Output devices, which translate the computer's output into a form understandable by the human being.

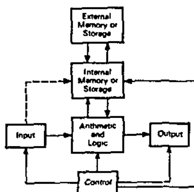


Fig 2. Components of a Digital Computer

Arithmetic Unit

4.2 The Arithmetic unit, or Accumulator, gathers information, and under the computer's control, acts on this data which it has either stored itself or which it recalls from the memory unit. For example, a number from the memory can be placed in the accumulator. Upon receipt of an order to add, a number will be added to the one already in the accumulator which then contains the sum Depending on the problem, this sum can be transferred to the memory, or be retained by the accumulator for further operations. In the case of a negative number, the accumulator will recognise the negative sign. For a "subtract" operation, the accumulator finds the difference and attaches the proper sign. To "multiply," a series of additions are made in the accum-

ulator, and to "divide", a series of subtractions. The series, or sequence, of operations is a programme, and the individual steps are *instructions*.

The Memory

4.3 The memory or storage unit, has a large number of individual locations or addresses, each capable of storing information till needed. Each location can be described and located, so that either its information or the new location where the information is stored can be obtained. A specific group of addresses is called a *Register*. Reading out a piece of information from the memory does not destroy the data--the same information can be referred to indefinitely. In a digital computer, all the information needed for solving a problem can be stored in the memory, including all the steps or the programme required for the solution. Once proper information has been placed in the memory, the computer is independent of all outside devices until a solution is reached. At this time, the final results are stored until needed, and then presented through the output device.

The Control Section

4.4 The control section directs the operation of the computer in order to fulfil the conditions set forth by the programme. The control unit draws the instructions and plans their proper execution by following the principle of operation for a given machine. For example, this selection may translate a "multiply" order into a series of additions (which it actually is to the computer).

Input and Output

4.5 Input and output devices are similar in operation, but perform opposite functions. An input unit reads information from punched cards, magnetic tapes, floppies, magnetic disks or keyboards. It also codes the data so that the computer can handle and use the information. The opposite function is performed by the output unit. It converts the computer results into a form usable by the operator, such as typed sheets, or punched cards for the control of production machines. Incidentally, magnetic devices are both input and output ones.

Questions

1. *Define an electronic computer.*
2. Compare the three types of computers, namely--digital, analogue and hybrid.

3. Describe briefly, the functions of a digital computer, with a block diagram.
4. Tabulate the characteristics of various generations of computers.
5. How would you classify digital computers ? What are their characteristics ?

2

Binary Arithmetic

1. Introduction

1.1 The computer's storage has been shown to store data for processing, and give output information when needed. Data is stored inside the computer in the form of 1s and 0s since most of the electronic devices operate in two stable states. This system of representing numbers as functions of 1s or 0s is called the binary system. But we are all familiar with decimal system where the number of states possible is 10 (O through 9). We are also familiar with the arithmetic operations of addition, subtraction, multiplication and division in decimal system because our forefathers could do arithmetic only with the ten fingers in their hands. As we shall see, we could operate with binary, ternary, octal and other forms of binary representation. While an executive user of a computer may make use of the computer even without an in-depth knowledge of binary arithmetic, an overall knowledge of the binary system is considered appropriate for him.

2. Weights or Radix

2.1 In the case of the decimal system, the factor by which the value ("weight") increases as we move leftward in a number called "radix" or "base", is 10. Considering an example, if we take a number 9847 in decimal notation, it really represents,

$$9 \times 1000 + 8 \times 100 + 4 \times 10 + 7 \times 1$$

As we can see from above, because of its position, 9 is being multiplied by a factor 1000 while 7 is multiplied only by 1. The same could be rewritten as

$$9 \times 10^3 + 8 \times 10^2 + 4 \times 10^1 + 7 \times 10^0 \quad \left(1 = \frac{10}{10} = 10^0\right)$$

Note : The value resulting from raising a number to a power (i e) the value resulting from exponentiation, is a product of successive multiplications by the number Thus,

$10^3 = 10 \times 10 \times 10 = 1000$

The figure 1 is obtained by dividing any number by itself and not necessarily only 10/10 . Similarly, 23 in decimal notation would be $2 \times 10^1+3 \times 10^0$

Binary Representation

2.2 From the above discussion, we see that each digit written in the decimal notation is interpreted as having a value equal to the absolute value of the digit (0,1,2,3 etc.) times the value of the position it occupies. The same is true for a binary system as well.

Incidentally, binary digits are also called "bits". Let us take a number 14 in decimal notation. In order to distinguish the 'base' or 'radix' of a number, let us write the radix as a subscript below the least significant digit --

$$\text{Thus } (14)_{10} = 8+4+2=(2 \times 2 \times 2)+(2 \times 2)+(2 \times 1)$$
$$= 2^3+2^2+2^1 = 1 \times 2^3+1 \times 2^2+1 \times 2^1+0 \times 2^0$$
$$= (1110)_2$$

Thus 14 in decimal notation = 1110 in binary notation.

Radix Conversion

2.3 In the above paragraph we saw that we could represent $(14)_{10}$ as $(1110)_2$. The conversion of a number say $(2875)_{10}$ to its equivalent binary number would be cumbersome. There is an easier method, however, which is described below. Whenever we wish to convert a decimal number into its binary equivalent, the following procedure is suggested·

Let us take a number $(53)^{10}$ as an example. First 53 is divided by 2 to give an integer quotient of 26 and a remainder of 1. The quotient is again divided by 2 to give a new quotient and remainder. This process is continued until the integer quotient becomes 0. The coefficients of the desired binary number are obtained from the remainders as follows:

Integer Quotient		Remainder	Coefficient
$\frac{53}{2}$ =	26	1	$C_0 = 1$
$\frac{26}{2}$ =	13	0	$C_1 = 0$
$\frac{13}{2}$ =	6	1	$C_2 = 1$
$\frac{6}{2}$ =	3	0	$C_3 = 0$
$\frac{3}{2}$ =	1	1	$C_4 = 1$
$\frac{1}{2}$ =	0	1	$C_5 = 1$

Thus, $(53)_{10} = (C_5C_4C_3C_2C_1C_0)_2 = (110101)_2$

The arithmetic operation can be performed more conveniently as follows:

Integer	Remainder
53	
26	1
13	0
6	1
3	0
1	1
0	1

Answer $=(110101)_2$

The conversion from decimal integers to any base r system is identical to the above example except that the division is done by r instead of 2.

Example: Convert $(569)_{10}$ to octal. In octal representation, the coefficients can have values (0,1,2...6,7) only.

Dividing by 8, we get

Integer	Remainder
569	
71	1
8	7
1	0

Answer $(1071)_8$

Rechecking, we find that

$$(1071)_8 = 1 \times 8^3 + 0 \times 8^2 + 7 \times 8^1 + 1 \times 8^0$$
$$= 1 \times 8 \times 8 \times 8 + 0 + 7 \times 8 + 1$$
$$= 512 + 56 + 1 = (569)_{10}$$

3. Fractions

3.1 There are many situations when one has to use fractions. Fractions like 1/2, or 2.563 where one could conveniently seek recourse to decimal point is common in nature. The conversion of a decimal fraction to binary is somewhat similar to the method used for integers. However, multiplication is used instead of division, and integers are accumulated instead of remainders. Let us take an example.

Convert $(0.8265)_{10}$ to binary. First, 0.8265 is multiplied by 2 to give an integer and a fraction. The new fraction is multiplied by 2 to give a new integer and a new fraction. This process is repeated until the fraction

becomes zero or until the number of digits has sufficient accuracy. The coefficients of the binary fraction are obtained from the integers as follows:

		Integer	*Fraction*	*Coefficient*
0.8265 × 2	=	1	0.6530	$f_1 = 1$
0.6530 × 2	=	1	0.3060	$f_2 = 1$
0.3060 × 2	=	0	0.6120	$f_3 = 0$
0.6120 × 2	=	1	0.2240	$f_4 = 1$
0.2240 × 2	=	0	0.4480	$f_5 = 0$
0.4480 × 2	=	0	0.8960	$f_6 = 0$
0.8960 × 2	=	1	0.7920	$f_7 = 1$
0.7920 × 2	=	1	0.5840	$f_8 = 1$
0.5840 × 2	=	1	0.1680	$f_9 = 1$

$(0.8265)_{10} = 0.f_1f_2f_3f_4 \ldots\ldots)_2$
$= (0.110100111)_2$

Rounding

3.2 In the above example, we could have continued with our arithmetic operation till we reached an absolute value. Unfortunately, there are fractions, (like the one shown above), and in fact, sometimes worse, where the fraction turns out to be a recurring one like 1/3 where the fraction is 0.333 ..... and so on. Unfortunately, each digit in a computer has to be accommodated and hence one has to strike a balance as to when one should stop. Such stopping poses a certain degree of degradation may be, but then are we justified in being "Penny wise and Pound foolish"?

3.3 The decimal fraction 2/3 is 0.1010... ad infinitum in the case of binary notation. The number of bits can be reduced to two by chopping off, or *truncating*, all bits after the second, so that 2/3 is approximately $(0.10)_2$. However, it is better to choose the closest available value, in this case 3/4 or $(0.11)_2$ since the error is then -1/12 rather than 1/6. Incidentally, $(0.10)_2$ = (0.5 × 1+0)=0.5 and $(0.11)_2$=0.5+0.25=0.75). Taking the closest value is called *rounding*. The error introduced is called *roundoff* error. By rounding, rather than truncating, the *roundoff* error can be kept to within limits, like in rounding, $(0.5468)_{10}$ would be written as $(0.547)_{10}$ in case we rounded to three decimal position, while it would be $(0.546)_{10}$ if we truncated the same. It is important to remember that integer arithmetic is exact as long as the range is not exceeded, whereas fractional arithmetic is only exact in those unusual cases where the numbers represented are exact. Generally, fractions will be in error by a small amount.

4. Overflow

4.1 Because representable numbers have only a finite range, arithmetic operations will give incorrect answers if the result is not in range. The error can be of the following types:

(a) If the answer is larger than the largest representable number, then the answer supplied by the machine will be grossly in error and we say that *overflow* has occured.

(b) The answer may also be between two representable numbers, so that the machine can choose one or theother as answer. This is the other case of roundoff error.

5. Floating Point

5.1 In the number representations discussed thus far, the computer would be insensitive to the location of the binary point. For example, suppose $(1011)_2$ is the number, if the decimal point is after the fourth digit, it would stand for $(11)_{10}$ whereas if the decimal point was before the first digit, it would mean $(0.6875)_{10}$. It is also not always convenient to restrict the representable numbers to either integers or fractions. In some problems, numbers which vary in size from 10^{10} to 10^{-10} may have to be used. For this reason, floating point notation was introduced in computers. In scientific notations, it is usual to write down a number as a digit between one and nine, preceded by a decimal point and followed by some more digits. Then, multiply this by ten to an appropriate power. The first part is called the mantissa and the latter the exponent. Thus,

$$123 = 0.123 \times 10^3$$
$$4652 = 0.4652 \times 10^4$$
$$19.832 = 0.19832 \times 10^2 \text{ etc.}$$

In this case, 0.123, 0.4652 etc., are called mantissa while 3, 4 and 2 are called the exponent (i.e) the power of 10. In a binary machine, however, we usually make use of an exponent base which is a power of 2 rather than 10. In a digital computer, part of the field reserved for storing numbers is used to represent mantissa while another part could be used to denote the exponent. In floating-point representation, round-off error in representing a number depends on the exponent .

5.2 Arithmetic operations on floating-point numbers can cause overflow if the resulting exponent is too large. The exponent may also be less than the most negative representable number, in which case the answer is very small Such a situation is called *underflow*.

6. Fixed Point

6.1 In the case of floating point representation, we could have denoted 0.005673 as

0.0005673×10^1
or 0.005673×10^0
or 0.05673×10^{-1}
or 0.5673×10^{-2}

Such a representation as shown in the fourth form, (0.5673×10^{-2}) where the value of the exponent is minimum, is called normalisation.

6.2 But the same number 0.005673 could have been denoted as 005673 and the user permitted to fix the decimal point. Such a representation of data internal to computers in which the decimal point is implicit is called fixed- point representation.

6.3 Having seen the types of number representation in digial computers, let us discuss the way in which we can perform arithmetic operations of addition, subtraction, multiplication and division using binary arithmetic.

7. Arithmetic Operations

Binary Addition

7.1 Arithmetic operations with decimal numbers depend on several rules which are usually learned at such an early age that the process appears "very natural" rather than being dependent on a set of rules and tables.

7.2 Almost all arithmetic operations in computers are performed through addition. In decimal addition, one number is placed under another number and the two are added. We carry a number from one column to the next when the sum of the first column exceeds 9. In binary addition, we carry a number from one column to the next when the sum of the first column exceeds 1. Let us now add two numbers in binary and see how it is done:

	Decimal	*Binary*
Addend	14	1110
Augend	+ 8	+ 1000
Carry	10	10000
Sum	22	10110

$= (16+0+4+2+0)_{10} = (22)_{10}$

In the case of binary, two 1s added together would be equal to 0 with a carry of 1.

From the above, a few rules for binary addition can be framed. These are

1 + 1 = 0 and carry 1 to add to the next column
1 + 0 = 1
0 + 1 = 3
0 + 0 = 0

Binary Subtraction

7.3 In the case of subtration in the binary system, one could follow the same procedure as applied for decimal notation, namely

	Decimal'	*Binary*
Minuend	13	1101
Subtrahend	-9	-1001
	4	$(0100)_2 = (4)_{10}$

The rules for binary subtraction would be *1--1=0; 1-- 0=1; 0--1=1* with borrow from the next column of the minuend, *0--0=0.*

Complement Method for Binary Subtraction

7.4 Let us digress a little and study the complement method and see if it could be of some use in our subtraction process. It would be appropriate to see the decimal complement method first and [illegible] system later. The tens [illegible] added to that numb[illegible] power of 10. For e[illegible], the tens complement of 3 is 7(i.e.) (10--3=7). [illegible] would be an appropriate Likewise, the complement of 25 is 75(100-25) and so on. Suppose we wish to subtract 3 from 8, we would proceed as follows:

(a) *Step 1* : Complement the subtrahend (viz.) 3, and it is 7
(b) *Step 2* : Add the minuend 8, and the complement 7.

Since the sum results in a carry in the high-order position, replace the carry by a + sign. Hence 8--3 = 5 Thus

8	
+7	
15	the difference of 8 -- 3 then is
= +5	equal to +5.

7.5 In case there is no carry, then the remainder is in complement form, and the final answer will be negative. When the remainder is in complement form, it is not the true remainder. To get the true remainder, it must be recomplemented. For example,

```
                 5
(in normal    -- 9
 arithmetic)  -------
              -  4
              -------
```

In the complementary method, the following steps would follow:

(a) Step 1: Complement the subtrahend, 9. The tens complement of 9 is 1.

(b) Step 2: Add 5 to the complement 1.

```
Hence      5
           1
         -------
         + 6
         -------
```

(c) *Step 3:* Since there is no carry in the high-order position, the remainder 6 is in complement form. The tens complement of 6 is 4, and its sign is negative. Hence the answer is --4

The Decimal Nines Complement Method

7.6 In the *nines* complement method, the complement of the subtrahend is the amount that must be added to that number to make a total of 9. That is, each decimal digit in the subtrahend is first subtracted from 9. For example, the complment of 2 is 7, the complement of 4 is 5, the complement of 35 is 64(99--35), and so on. Let us take an example.

```
Minuend         79
Subtrahend    -- 45
Difference    + 34
              -----
```

The various steps are:

(a) *Step 1*· Take the Ninety nine complement of the subtrahend: (i.e.) Complement of 45 = (99--45)=54.

(b) *Step 2:* Add the complement to the minuend

Minuend	
(Ninety nine	79
complement)	+ 54
The sum is	133
	1
	+ 34

(c) *Step 3:* Since there is a carry in the high-order position of the sum, it is added to the units digit and the total value is given a plus sign. Hence, the remainder of 79--45=+34.

7.8 If no carry develops, however, the sum is in complement form and is negative also. A recomplement is necessary to get the true remainder. For example,

Minuend	
Subtrahend	47
	-- 79
Difference	
	-- 32

The various steps are

(a) *Step 1:* Take the Ninety-nine complement of the subtrahend: Ninety nine complement of 79 is 20(i.e. 99-79=20).

(b) *Step 2:* Add the complement to the minuend.

Minuend	
Ninety-nine	47
complement	+ 20
The sum	
	+ 67

(c) *Step 3:* Since no carry develops, recomplement the sum and add a minus sign to the true remainder. The nines complement of 67 is 32 (99--67=32). The true remainder, therefore, is --32.

Binary Subtraction by Complement Method

7.9 From the foregoing discussions, one might wonder as to whether all the trouble of determining of the complement is necessary after all. But in case of binary system, one could go in for one's complement or a two's complement. Let us consider some examples. Ones complement is obtained by reversing each digit to its opposite state (say, 1 will become 0 and 0 will become 1). Thus, ones complement of 1011 would be. 0100.

For example
if we wish to subtract 2 from 8.then

	Decimal	Binary
Minuend	8	1000
Subtrahend in	-- 2	-- 0010
Difference	+ 6	$(+0110)_2$

(b) *Step 2:* Since a carry has developed in the high-order position, it is added to the units position and the result is given a plus sign:

```
  1000
+ 1101
------
 10101
     1
------
```

$(+0110)_2 = (+6)_{10}$

Thus, the result is +6.

7.10 If no carry develops, however, the result is negative remainder in a complement form. It has to be recomplemented and a negative sign added to get the true answer. For example,

	Decimal	Binary
Minuend	8	.1000
Subtrahend	-- 12	-- 1100
	-- 4	-- 0100

The various steps are as follows:

(a) *Step 1*: Complement the subtrahend by reversing its bits, and then add:

```
  1000
+ 0011
------
  1011
------
```

Since no carry develops, the result is recomplemented to give 0100 and given a negative sign. A point to note is that, in binary subtraction, both the minuend and the subtrahend must have equal length.

Binary Subtraction -- Two's Complement Method

7.11 This method of binary subtraction is performed by taking the one's complement of the subtrahend, then adding 1 to the low-order digit. Any end-carry is replaced by a sign. If no carry occurs, the result is recomplemented and given a minus sign. This is best illustrated by means of an example. Suppose, we wish to subtract 0011 (3) from 1000 (8). One would proceed as follows:

(a) *Step 1*: Determine the one's complement of the subtrahend. Add 1 to the low-order digit

Subtrahend	0011
Ones complement of subtrahend	1100
Add 1	1
Two's complement	= 1101

(b) *Step 2*: Add Minuend and the two's complement

```
   1000
 + 1101
 ------
  10101
 ------
```

The circled digit is an end carry and is changed to a plus sign. Therefore, the true remainder becomes +0101 which is $(+5)_{10}$.

7.12 If no carry develops, however, the remainder must be recomplemented and given a negative sign. Let us try to subtract 7 from 2.

(a) *Step 1*: Determine the ones complement of the subtrahend.

Subtrahend	0111	$(7)_{10}$
One's complement	1000	
Add 1	1	
Two's complement	1001	

(b) *Step 2*: Add the minuend to the two's complement of the subtrahend.

Thus,

Minuend	$(2)_{10}$	0010
Two's complement of subtrahend		1001
		1011

Since there is no carry, we have to recomplement the remainder (reverse the digits and add 1 to the low-order digit) and add a negative sign.

Thus, the true remainder becomes,

$$-(0100+1) = -(0101)_2 = -(5)_{10}$$

Which Complement to Choose ?

7.13 There is no gainsaying the fact that the computer hardware designer would attempt to simplify the computer circuitry. Subtraction is usually performed by complementing the number to be subtracted and then adding the complement. This simplifies circuitry because subtraction can make use of the addition circuitry. Also complements are used to simplify subtraction operation and for logical manipulation. The change to the complement method of subtraction reduces all computer arithmetic to forms of addition. But the problem before us now is as to which complement to choose (Two's complement or one's complement) ?

7.14 A comparison between 1's and 2's complements reveals the pros and cons of each. The ones complement has the advantage of being easier to implement by digital components since the only thing that must be done is to change 0's to 1's and 1's to 0's. The implementation of the 2's complement may be obtained in two ways :

(a) by adding 1 to the least significant digit of the 1's complement, or

(b) by leaving all leading 0's in the least significant positions

and the first 1 unchanged, and only then changing all 1's into 0's and all 0's into 1's.

7.15 During subtraction of two numbers of complements, the 2's complement is advantageous in that only one arithmetic operation is required. The 1's complement, on the other hand, requires two arithmetic additions when an end around carry occurs. The 1's complement has the additional disadvantage of having two arithmetic zeros; one with all 0's and one with all 1's. To illustrate, let us consider the subtraction of two equal binary numbers say 0100--0100=0000.

Using 1's complement,

```
    0100
  + 1011       Complement again to
  ----------
  + 1111       obtain --0000
```

Using 2's Complement,

```
    0100
  + 1100
  ---------
   10000
=  + 0000
```

While the 2's complement has only one arithmetic zero, the 1's complement zero can be either positive or negative, which may complicate matters.

7.16 The 1's complement is also useful in logical manipulations since the change of 1's to 0's and vice versa is equivalent to a logical inversion operation. The 2's complement is used only in conjunction with arithmetic applications. Consequently, it is convenient to adopt the following convention. When the word complement, without mention of the type, is used in conjunction with a non-arithmetic application, the tvne is assumed to be the 1's complement.

Binary Multiplication

7.17 The binary multiplication table involves four basic steps :

$0 \times 0 = 0$

1 × 0 = 0
0 × 1 = 0
1 × 1 = 1

In the binary mode, multiplication is done by one of the two methods–

(a) the decimal method or
(b) the shift method.

The Decimal Method

7.18 This method usually utilizes the technique we follow when we multiply manually using pencil and paper (i.e.) we multply by each bit of the multiplier and then add. For example,

	Decimal	Binary
Multiplicand	4	100
		× 011
Multiplier	× 3	
		100
Product	+ 12	100
		1100= $(12)_{10}$

The Shift Method

7.19 In this method, multiplication is performed by multiplying the multiplicand by the first left hand bit of the multiplier and then shifting left one position and adding 0. The same procedure follows on the second-position bit of the multiplier, and so on until the operation is completed. For example,

	Decimal	Binary
Multiplicand	2	0010
	×	×
Multiplier	3	0011
Product	+ 6	0110

The various steps are as follows :-

(a) *Step 1:* Multiply the multiplicand by the first left-hand bit of the multiplier.

```
10
11
---
10
```

Product of the first multiplier

(b) *Step 2 :* Shift left one by adding 0 to the right. The product (10) of step 1 becomes 100.

(c) *Step 3 :* Multiply the multiplicand by the next bit of the multiplier.

```
10
11
---
10  Product of the next multiplier.
```

(d) *Step 4 :* Add the product obtained in step 3 to the shifted product in step 2. (i.e.)

```
100
+10
----
110  The final product.
```

Multiplication on a computer is performed in a similar way.

7.20 There are situations (while multiplying) that give rise to problems of adding more than two 1's. For example,

```
    1 1 1 0    Addition handled as pairs
  × 1 1 1 1    (Consider Col. 3) (Col. 4) (Col. 5) (Col. 6)
  ---------         /1         /1      1        1
    1 1 1 0      1 (        1 (       1        1
  1 1 1 0           \1         \1
                               /1      1        1  From
1 1 1 0                      1(                    col. 4
                               \1      0        1  From
                                                   col. 5
1 1 1 0
---------------  ------   --------  ----   ------
1 1 0 1 0 0 1 0   10       100       11     100
C C C C C C       Add 1    (Add 1
O O O O O O       carry to carry to  Add 1   Add 1
                  col.4    col.6     to col.6 to col.8
```

1\. 1. 1. 1. 1. 1.
8 7 6 5 4 3
Hence the result is $(1\ 1\ 0\ 1\ 0\ 0\ 1\ 0)_2 = (210)_{10}$

7.21 Binary Multiplication does not need any pre-determined formulae really. If you wish to multiply any number by $(2)_{10}$ or $(10)_2$ all you have to do is to shift the number by one position to the left. For instance, if we wish to multiply $(0111)_2$ by $(0010)_2$ all you have to do is, to shift $(0111)_2$ by *one* position to the left (i.e.) $(0111)_2$ multiplied by $(0010)_2$ becomes $(1110)_2$. In other words, by multiplying $(7)_{10}$ by $(2)_{10}$ we get $(14)_{10}$. In a similar vein, if we wish to multiply a number by $(4)_{10}$ or $(0100)_2$, all we need to do is to shift the binary representation of the number by *two* positions to the left.

Multiplication of fractions is also easy. For example, $(0001.1)_2$ when multiplied by $(0010)_2$ is equal to $(0011.0)_2$ which is equal to $(3)_{10}$.

Binary Division

7.22 In both the decimal and binary systems, division is performed by successive subtraction. It is, in fact, the opposite of multiplication, which is a series of additions. When we divide, the quotient is the number of times the divisor is subtracted from the dividend. For example,

16÷4
16–4=12 Result of 1 subtraction
12–4=8 " " 2 "
8–4=4 " " 3 "
4–4=0 " " 4 "

The quotient is, therefore, 4.
In binary, the same procedure is followed. For example,

```
        0100          (4)
        -----
100     10000
        100
        -----
        000
```

7.23 Let us take another example. Let us divide 330 by 6. Example:-

```
     55
   -----
6) 330
   30
   ---
    30
    30
    --
     0
```

In the case of binary division,

```
           01011
        ---------------
110        1000010
           110          -- Divisor greater than 100, so put 0 in quotient
        ------------
           1000         -- Add digit from dividend to group used above
            110         -- Subtraction possible, hence put 1 in quotient
        ------------
            100         -- Remainder from subtraction plus digit from
                           dividend

            110         -- Divisor greater, so put 0 in quotient
        ------------
            1001        -- Add digit from dividend to group

             110        -- Subtraction possible, so put 1 in quotient
        -------------
             110        -- Add digit from dividend to remainder
             110        -- Subtraction possible, so put 1 in quotient
        -------------
               0        -- No remainder, so stop
```

7.24 The above steps can be summarised as follows:-

- (a) Start from the left on the dividend.
- (b) Perform a series of subtractions in which the divisor is subtracted from the dividend.
- (c) If subtraction is possible, put a 1 in the quotient and subtract the divisor from the corresponding digits in the dividend.
- (d) If subtraction is not possible, (divisor greater than remainder) record a zero in the quotient.
- (e) Bring down the next digit to add to the remainder digits. Proceed as before in a manner similar to long division.

7.25 Binary division, just as in the case of binary multiplication, is a very simple operation indeed. For example, if we wish to divide $(8)_{10}$ or $(1000)_2$, by $(0010)_2$, all we have to do is to shift the numerator by *one* position to the right thus giving rise to $(0100)_2$ as the answer. In a similar vein, division by $(0100)_2$ would entail shifting the numerator two positions to the right.

Even fractions can be obtained with ease. Suppose we wish to obtain the result of

$$\frac{(0001.00)_2}{(0010)_2}$$

The answer is $(0000.10)_2$ which is $(0.5)_{10}$

This has been obtained without remembering any rules.

8. Negative Numbers

8.1 In the above paragraphs, we have discussed the various methods of representing positive integers. Let us now see as to how negative numbers could be represented. In conventional arithmetic, we prefix a minus sign to negative numbers and occasionally put a plus sign in front of positive numbers when there is likely to be some ambiguity. This is called a "sign and magnitude" representation. First, one finds out whether a number is positive or negative and then how large it is. This method has been used in several computer designs, usually with one bit reserved for the sign position, zero meaning plus and one meaning minus. A four is a four and has the same bit pattern regardless of whether it is plus four or minus four. Only the sign bit will be different.

Questions

1. Define a radix.
2. Convert the following into binary :
   (a) 23
   (b) 63
   (c) 4095
   (d) 11
3. Convert the following into binary :
   (a) 0.125
   (b) 0.563
   (c) 0 097
4. Convert the following into decimal notation :
   (a) $(10111)_2$
   (b) $(0.01101)_2$
   (c) $(1011001)_2$
5. Differentiate between rounding and truncation.
6. Distinguish between floating point and fixed point.
7. Add the following using binary notation :
   (a) 7 and 11

8. Perform subtraction on the numbers given below using 2's complement :
   (a) 7 minus 4
   (b) 4 minus 9
   (c) 15 minus 6
9. Perform subtraction using 1's complement :
   (a) 9 minus 5
   (b) 3 minus 11
   (c) 17 minus 30
10. Using the shift method, multiply the following numbers after converting them to binary :
    (a) 5 by 4
    (b) 13 by 9
    (c) 21 by 31
11. Divide after conversion to binary, the following numbers :
    (a) 11 by 2
    (b) 9 by 3
    (c) 32 by 4
12. Illustrate, with an example, how all arithmetic operations can be reduced to one of addition.

3

Internal Operation of a computer

1. Introduction

1.1 Normally, the digital computer stores data and instructions in a binary form. Such a representation involves putting binary digits together. There are other methods like Binary coded Decimal, Extended Binary-coded Decimal etc. which are also used in storing alphanumeric data. Once the data and instructions are stored, it is necessary to manipulate the data to derive the desired results. It has been seen in Chapter I that the Central Processing Unit is the main component of the computer. The manipulation chores outlined above are performed by the Central Processing Unit (CPU). Right from an abacus to the present day computers, some control on the activities of various units of the computer system is required to be exercised. Such control would involve sequencing of instructions, moving them at the appropriate time periods to various circuits, interpreting, and executing the commands of instructions.
1.2 Though the subject of internal operations of the computer is a complex one, a certain rudimentary knowledge of the functioning of various components would be appropriate at this juncture. It is also worth the while to distiguish the types of *ware* associated with the computer systems

Types of Ware

1.3 The various types of ware one comes across in computer technology are as follows:

(a) *Hardware:* The physical units associated with a computer system are called Hardware. Hardware also involves equipment which can perform the following functions:
   (i) Data preparation.
   (ii) Input to the computer.
   (iii) Processing.
   (iv) Secondary or auxiliary storage.
   (v) Output from the computer.

(b) *Software:* This denotes the various computer programmes (both

user-oriented and system-based). Utility programmes which are application-oriented and the various system programmes also fall under this category.

(c) *Firmware*: In modern computers, a new term called "Firmware" has been brought into play. This is an intermediate step between hardware and software. Rather than wiring certain functions, a manufacturer may use a special storage to store routines (which perform the functions) by calling upon sequences of elementary hardware functions. The user normally is not aware of the existence of firmware because the computer accepts instructions and then executes them either through the logic already stored or directly through the hardware.

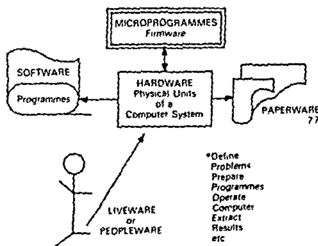


Fig. 1 Inter-relationships Between Different Wares

(d) *Liveware* : The people associated with computer systems are euphemistically called "Liveware".

1.4 A pictorial representation of the various types of ware showing their inter-relationships is given in Fig. 1.

1.5 While "Liveware" is an obvious concept, "Firmware" is beyond the scope of this discussion. "Software" will be described in the next chapter. In this chapter, it is proposed to analyse the various facets of the internal operations in a digital computer using "Hardware" under the following headings:

(a) Internal Number Representation.

(b) Symbolic Logic.
(c) The Central Processor and Addressing Systems.
(d) Addressing and Instruction Format.
(e) Computer Operating Cycles.

2. Internal Number Representation

Binary-coded Decimal

2.1 The representation of data inside a computer could best be achieved in a binary form. There is another approach to bit representation which is used in many business-oriented computers called the Binary-coded Decimal system. This system envisages use of individual bit sets for every decimal digit. Straight binary representation normally provides faster operating speeds and is, therefore, emphasized in scientific computers (where internal computation speeds are important). Commercial applications, on the other hand, *typically involve much input and output but very little computation.* Therefore, the business computer is oriented towards fast conversion from, and, to decimal form, and ease of programming and data handling. The use of Binary-coded Decimal or the related binary-coded character codes can simplify the input/output conversion and input/output programming. This, however, occurs at the expense of internal operating speeds.

2.2 The Binary-coded Decimal form uses a separate set of four binary digits to represent each decimal digit. The four bit positions are interpreted as in straight binary. This coding, often termed "Natural binary-coded decimal," is the most common code, but variations of this have also been used. Fig. 2 shows the various combinations used in the Binary coded Decimal.

4 bit set	Decimal value	4 bit set	Decimal value
0000	0	0110	6
0001	1	0111	7
0010	2	1000	8
0011	3	1001	9
0100	4		
0101	5		

Fig. 2. Permitted Combinations of Binary-coded Decimal

2.3 Even though 16 decimal digits can be represented in a 4 bit combination, we can use only 10 of these to represent decimal digits. Binary-coded arithmetic is slightly different from binary arithmetic. The rules provide for a decimal-type carry when the results from adding two digits

computer design, it is possible that some errors might creep in or some bits might be lost during transfer of data. Such problems are solved by the introduction of an error-control mechanism called the parity bit or binary check bit. The check bit is a binary digit which is added to the bit configuration representating data in a specific computer. A four-bit code becomes five bits, a six-bit alphanumeric code becomes seven bits, and an eight-bit alphanumeric code will have a total of nine bits. The error control mechanism counts the 1 bits of the character-bit set and adds a check bit to the sum to make the total number of 1 bits odd or even, depending on the computer design In computers working with odd parity, a check bit (1 bit) is added to the character-bit set when the total number of 1 bits is even. Otherwise, 0 check bit is recorded. In computers specifying even parity, a check bit is attached when the total number of 1 bit is odd. Otherwise, 0 check bit is recorded.

2.7 Having seen the different numbering schemes, let us turn our attention to the Logic used in computers.

3. Symbolic Logic

3.1 Computers, despite their complexity, are composed of surprisingly few types of circuits. Though circuit elements and circuit designs vary from machine to machine, the computer proper consists basically of two types of *functional elements: storage elements and logic circuits. While storage* elements retain information (number, instructions, signals, codes etc.) for longer or shorter periods of time, logic circuits perform logic (or arithmetic) operations. They are also referred to as decision circuits, switching circuits or logical elements. The action of logic circuits can be described well by symbolic logic. In turn, logic circuits may be considered to be electronic implementations of Boolean algebra.

Boolean Algebra

3.2 Mathematical or Symbolic Logic is frequently referred to as Boolean algebra in honour of George Boole, one of its early contributors. Though the original work of arriving at a "general method in which all truths of reason would be reduced to a kind of calculation" was initiated by G. W. Leibnitz, it was Boole who gave the world the form of logic now known as Boolean Algebra. Before we embark upon the mechanics of the symbolic logic, let us venture a few remarks on logic itself.

Introduction to Logic

3.3 Logic is the science of establishing the validity of thought of reason,

so that what is true in one statement will be true in all equivalent statements. Truths can be classified into four statements, called propositions such as -

(a) All A is B ("All dogs are animals"),
(b) Some A is B ("Some dogs are Labradors"),
(c) No A is B ("No dogs are birds"),
(d) Some A is not B ("Some dogs are not Labradors").

3.4 The above four absurdly simple examples are recognised truths. Working from them, and from them alone, it is possible to deduce a number of further truths. For example, let us change (a) into (b), (c) and (d) and see which are true and which are obviously false--

(a) all dogs are animals
(b) some dogs are animals
(c) no dogs are animals
(d) some dogs are not animals.

3.5 Assuming (a) is true, then (b) is obviously true always; but (c) and (d) are obviously false always. Many other deductions can be made from these four statements. Propositions, if true, can be put together to form a Syllogism: if it is true that: all dogs are animals, and that Rover is a dog, then one can deduce without a doubt that Rover is an animal.

Logical Connections

In logic, two connections --AND and OR -- express the relationships between two statements. *Or* means either class A or class B individually, or both class A and B together. For example, the class "Females" and the class "piano players" can be thought of individually, or the two can be thought of as two groups--say, in two separate rooms, or mixed together in one large room. Let us take another example to illustrate the use of connections--

(a) Kapoor is happy or he is rich
(b) Kapoor is happy and he is rich.

If, suppose, we represent "he is happy" by A and "he is rich" by B, then the first statement can be symbolically represented as $A \vee B$ (A or B) and the second by $A \wedge B$ (A and B). Thus we can write a table called Truth Table which would summarize the different combinations of truth values which may apply to the individual elements of a statement and show the resulting truth values for the statement. The truth values for $A \wedge B$ and $A \vee B$ are shown in Fig 4, with true being represented by 1 and false by 0. The

statement $A \vee B$ (Kapoor is happy or he is rich) is false only if both "he is happy" and "he is rich" are false, while the statement $A \wedge B$ (Kapoor is happy and he is rich) is false if either of the elements is false.

All combinations of Truth values for elements		*Truth value for Statements*	
A	B	$A \wedge B$	$A \vee B$
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Fig. 4. Truth Table

3.7 The Boolean Analysis can be applied to a network of switches. If a switch is open, it can be described as false, and true if it is closed. A network is true if current flows through it to the output line. Therefore, two switches A and B which operate in series such that there is current at the output line C only if both switches are closed is described in Boolean terms as $A \wedge B$.

Logic Circuits

3 8 The circuitry of a computer is designed from combinations of rather simple AND, OR, NOT AND, NOT, AND NOT OR logic modules. The basic circuits are combined in complex groupings to perform the switching for computer operation logic. The simple logic circuits of AND, OR and NOT are represented in Fig. 5. The AND gate (also called & gate, $\wedge$ gate and AND block) can be thought of as an electronic element into which two (or more) wires enter and one wire exits. The logic statement $A \wedge B$ (A and B) is represented by the AND gate: i e., for a pulse to come out on the exit wire, pulses must be sent into the element on both input wires. The truth table for $A \wedge B$ presents the operating characteristics of the AND gate when 1 is used to represent a pulse, and 0 a non-pulse.

3.9 The OR gate (also called V gate, OR block and mixer) represents the logic statement $A \vee B$ (A OR B). Like the AND gate it has two (or more) input wires and one output wire. However, the OR gate will allow an output pulse if either or both of the input lines receive an electrical pulse. The truth table $A \vee B$ specifies the operating characteristics of the OR gate

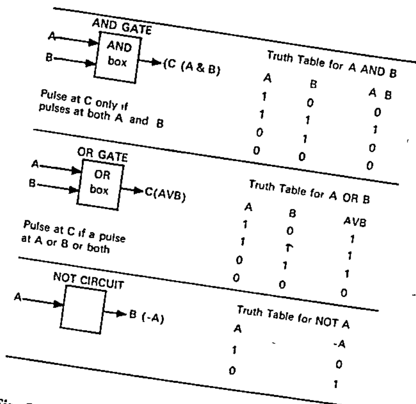


Fig. 5. Truth Table for AND, OR and NOT CIRCUITS

3.10 In the NOT circuit, the output is opposite of the input. NOT is also represented by-- A and $\bar{A}$. In the design of computers, it is desirable to standardize as much as possible. If a single logic module could be used for forming any logical function, this would simplify production. Two common modules are the NAND circuit (NOT AND) and the NOR circuit (NOT OR). The inputs for these two are the same as AND and OR respectively and the outputs are just the opposite of AND and OR.

3.11 With this preliminary introduction, let us turn our attention to the various facets of the core of the computer, the Central Processor.

4. The Central Processor

4.1 A computer must be able to perform certain fundamental operations internally. These can be classified as ability to--

(a) represent data and instructions,

(b) hold (store) data and instructions,
(c) move the data and instructions internally,
(d) interpret and execute the commands of the instructions.

4.2 These internal operations are performed by the Central Processing Unit (CPU) of the computer. The CPU consists of three sub-units-- the control unit, the Processing or Arithmetic unit and the storage unit. As explained earlier, these three units plus the input and output devices form a simple computer system.

Operation of a Central Processor

4.3 There is no mechanical movement in the CPU such as one finds in a mechanical calculator or an adding machine. All operations are performed electronically. The operations are directed by a programme of instructions held in the storage or memory unit, in which the data to be operated upon is also held. However, the storage unit is passive in that the data and instructions are merely held there. To be acted upon, the data and instructions must be moved into the control and processing units. The information held in the storage unit is represented by the polarity in one of two directions of sets of cores or similar storage media. When data or instructions are moved from storage to the processing or control unit, the information being moved is represented by the two electrical states: *pulse* and *no pulse.* While being operated upon in the processing or control units, the data and instructions are held in registers which are devices designed to accept electrical pulse representations, temporarily hold the data, and then release it in the form of electrical pulses when directed by control circuits. Computer operations such as comparison, arithmetic etc., are performed by groups of logic (switching) circuits. This suggests that much of the computer circuitry is for the purpose of controlling and counting the movement of the electrical pulses. The relationship between storage and CPU circuits is shown in Fig. 6. The circuits of the control and arithmetic units perform two basic functions: (1) control the flow of electrical pulses and (2) provide temporary bit storage.

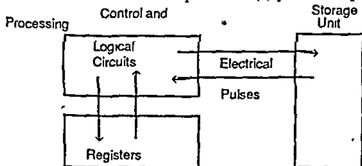


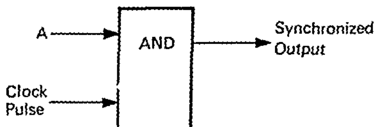
Fig. 6. Relationship Between Storage and Registers

4.4 From the foregoing, one is led to believe that for the successful operation of a CPU, one must have a facility to--

(a) control the movement of pulses,
(b) store bit values for a short period of time, and
(c) perform arithmetic

Control of Movement

4.5 It takes time for electrical pulses to reach the proper level for representing a bit and time for the pulse to travel through the computer. Also, operations in the computer must be performed in an orderly, and timed sequence. The control of timing is performed by the use of control signals called clock pulses which are emitted from a central device called the Master clock. These timing signals are sent to all of the logical components to keep them synchronized with each other. For example, let us assume a pulse A which must arrive at a point B at an exact time. By inserting an AND gate and a clock pulse, the pulse at A cannot pass until the clock pulse is also received.



Registers

4.6 One of the functional units used in a CPU is a register. These registers are used to receive, store and transfer information used by the processing unit. A register is an assembly of listable circuits. It acts as a temporary memory for the processing unit. There are several registers having different functions These are--

(a) Accumulator register to accumulate results.
(b) Multiplier-divider register to hold the multiplier for multiplication and divisor (or the quotient) for division
(c) Instruction register, to hold the instruction being executed.
(d) Address register, to hold the address of a storage location or device and
(e) Storage register to hold data taken from or being sent to storage.

Accumulator

4.7 While the majority of registers mentioned above are complex in nature

the Accumulator is simply a device used for addition. The design of the accumulator will depend on whether the computer uses a Binary or a Binary-coded Decimal code or whether the arithmetic is to be serial (one digit at a time) or parallel (all digits at once). The major difference between a serial and parallel arithmetic is in the number of adder units required. Serial addition uses only one 1-digit adder and adds all pairs simultaneously using a different adder for each pair of digits. The serial accumulator is slower than the parallel accumulator but requires less hardware.

Serial		*Parallel*	
10100		10100	
10111			
--------		10111	
1		--------	
10		00011	Sums
10			
0	· Result of adding a	11	Carry
10	pair at a time	--------	

101011		101011	Final sum

Fig. 7. Serial and Parallel Addition

4 8 Having seen some of the components of a CPU, let us see as to how the CPU (Accumulator and Registers in particular) can assist in solving simple problems.

5. Addressing and Instruction Format

What is an address ?

5.1 The various storage devices (which we shall see in greater detail later) are not usable unless data stored can be retrieved when needed. In order to accomplish this, each storage location (bit, byte or a combination of bytes to form a word) is given an address which identifies it just as a house address identifies a house. Each set of cores in a core storage unit has an address. The address identifies the location so that data may be stored there and data so stored may be retrieved. The address is, therefore, a code which identifies the location for the computer circuitry. Each address refers to a set of storage devices, but how large the set is and what can be stored there depends on the word structure of the particular computer.

5.2 Let us consider a simple problem which the computer should be able to do, and with its help study the various addressing schemes.

Four-address Machines

5.3 Any operation requires the following components:

(a) An instruction to indicate what to do.This could be in the form of a numeric code (say 01 may mean ADD). This code could also be symbolic in nature (i.e.) MPY for multiply.

(b) The operands (or their address) which are to be operated upon.

(c) Address of a location where the result is to be stored.

(d) Address of the instruction where the next instruction would be found.

5.4 In its simplest form, a four-address machine would be required. Here, "ADD A, B, C, D" would mean "add the contents of A to the contents of B and put the resulting sum in C. Find the next instruction in location D". This type of instruction format was used in many of the early machines where the programmer had to keep track of various addresses.

Three-address Machines

5.5 In order to overcome the necessity of stating the address where the next instruction would be found (viz., D in the above example) we could place instructions in successive cells excepting when a transfer (called Jump) is required from the normal sequence. Thus, a 3-address machine would perform the indicated binary or diadic operation (that is an operation on two operands) on the contents of locations X and Y and store the results in location Z. The instruction would be somewhat like ADD X, Y, Z.

Two-address Machines

5.6 The three-address structure is not normally used for cost reasons. If we were to programme an instruction of the type A = B+C+D+E, the three-address code would be

ADD B, C, A
ADD A, D, A
ADD A, E, A

All but the first instruction needed only two different addresses, but the programmer had to provide three. This is overcome in a two-address machine by providing two-address instructions. A two-address instruction has the form

ADD A, B

Which means "ADD the contents of A to those of B and store the result in A". The result could also be stored in B. It can be seen that the instruction A = B+C+D+E could be easily written as

MOVE A, B

computer operation. While a computer which uses fixed time intervals to determine when to initiate the next operation is known as synchronous, an asynchronous machine initiates the next operation when a signal from the current operation indicates a completion. In both cases, exact timing is necessary for proper operation. The basic timing interval is a pulse emitted by electronic clock circuits. Pulse times range upto several million pulses per second.

Cycles of Operation

6.2 A computer cycle can be divided into two parts--- the instruction cycle and the execution cycle.

(a) *The Instruction Cycle.*

The sequence is as follows:

(i) An instruction is obtained from a main storage location and transfered to the central processor. The instruction is composed of an operation code specifying what is to be done and a code specifying the adress of the data or device to be operated upon (operand).

(ii) The operation code is transferred to an instruction register and decoded.

(iii) The address of the operand (if a storage address) is transferred to an index register.

(iv) The address of the next instruction is determined.

(b) *Execution Cycle*

(i) If the operand refers to information stored in memory, the information is obtained from storage and placed in the storage register.

(ii) The operation specified by the operation code is performed, using appropriate registers.

7. Summary

We have seen that characters are represented in binary form internally in the computer and that we have the BCD code and the EBCDIC code for external usage. The Algebra as propounded by GEORGE BOOLE in his paper "Laws of Thought" is very useful in computer logic and arithmetic operations. The central processor with its control unit, Arithmetic unit and storage form the core of the computer.

4.

Storage Devices and Software

1. Introduction

1.1 One of the functional components of a computer system is the storage or memory. Memory is required so that instructions and data could be preserved and recalled when desired. Memory is needed in many parts of the computer system. The various types of memory are illustrated in Fig. 1.

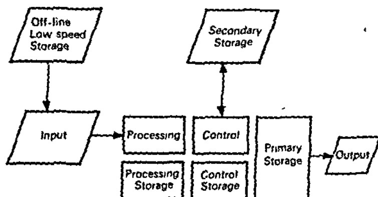


Fig. 1 Types of Storage

1.2 The main memory, or primary storage, is the storage associated with the CPU. Magnetic core was used for memory in the earlier generations. Most of the machines today use semi-conductor memory. Secondary or auxiliary storage is less expensive for holding data and programmes. Data and programmes to be used are read from secondary storage into primary storage for processing. The processing and control units of the CPU have devices such as registers for use as temporary processing storage. Another type of storage in some computers is a control unit storage which contains a computer programme for interpreting computer instructions. It is also

referred to as "Read-only storage", "stored logic" or "micro logic". In this discussion, let us confine our attention to the primary storage.

1.3 While we have primary storage to hold instructions and data, we must be able to tell the computer as to what it should do As described earlier, a computer is basically a moron which will do only what we tell it to do.

1.4 We found that the computer operates only with numbers and that too binary ones. It is, therefore, necessary for us to be able to communicate with the computer in the language it understands best, viz., Binary. Such a language is called machine language. The original programmers did exactly that. Later, as technology advanced, it was found to be very time-consuming and tiresome causing untold errors. Therefore, computer scientists developed what are called symbolic languages. In such languages, a symbolic code, MPY for multiplication, DVD for Divide etc., was used which could later be converted to machine-language instructions and used. Today, we have what are called high-level languages, as opposed to low-level ones of the earlier era, which permit users to make use of ENGLISH-type commands while instructing the computer. These commands, however, would be converted to machine-language before execution. These and other programmes are described under the term software.

1.5 It is, therefore, proposed to describe the two facets of importance in the computer operation under the following headings.

(a) Computer storage.
(b) Computer software.

2. Computer Storage

2.1 Primary storage, as mentioned earlier, consists of any device which will retain business, scientific and related information placed in it either temporarily or permanently until it is needed. It is also referred to as "working memory" because computations and other processing routines are "worked" in it. Data stored outside the computer system are usually kept in a file device. The external devices retaining such data are called file-storage devices. Therefore, external storage is called File storage or File memory.

Characteristics of Primary Storage

2.2 While one may assume that any storage device capable of being stable in two states should work as primary storage, there are certain characteristics that distinguish the *primary storage* from *secondary ones*. Some of these may be common with secondary storage but, by and large, they certain essential features. These are :

(a) *Immediate access to data stored in memory:* For the

and logic units to operate efficiently, the information stored in working memory should be directly accessible high speed.

(b) *Reusability:* It should be capable of erasing unwanted data and storing new data in its place.

(c) *Permanent recording of data already in storage:* The computer's primary storage unit should be designed to retain stored data intact in the event of electrical failure.

(d) *Automatic or self-checking ability:* In order to verify the accuracy of data represented in storage, a primary storage device should have an automatic self-checking feature, called parity check. The computer counts the number of bits of information in storage and upon the destruction or loss of any single bit, signals an error indication on the console.

(e) *Durability:*Unlike punched cards, which wear out in time, primary-storage devices are built to last permanently in spite of the constant storing and re-storing of data in them.

(f) *Compact size:* Because space is always at a premium, a primary storage device should be physically small, yet capable of storing a large volume of data.

Types of Primary Storage

2.3 There are many two-state devices which can be used to represent binary digits for computer storage as indicated in Fig. 2. The popular ones among them are:

(a) Magnetic core.

(b) Thin-film.

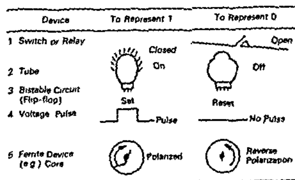


Fig.2 Types of Primary Storage

(c) Plated-wire.
(d) Semiconductors.
(e) Holographic memories.

Magnetic Core Storage

2.4 Magnetic core devices are very popular and are used in most commercial computers. They consist of a doughnut shaped ring of ferromagnetic material measuring about 1/16 inch on the outside diameter. Ferrò-denotes the presence of iron, and magnetic indicates the ability of the material to be magnetized. A magnetic core can be magnetized quickly and is highly retentive. Unless it is demagnetized, it is capable of retaining its magnetism almost indefinitely.

2.5 Because of the importance of speed, cores are made very small, which means faster switching and less magnetizing force required to change their status. The magnetizing force is generated by running a heavy current through two wires, usually called X and Y, passing through the core as shown in Fig. 3.

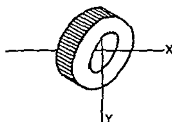


Fig. 3 Magnetic Core with X and Y Wires

2.6 Some of the advantages of magnetic cores are:
(a) Dependability.
(b) Durability.
(c) High-speed access.
(d) Low-cost operation.
(e) Large capacity.
(f) Low heat dissipation.

(g) Adaptation to word-oriented or character-oriented machines.

2.7 The magnetic core has only two definite states--the 0 state and the 1 state It can thus store a 1 bit or a 0 bit of information. If values greater than 1 are to be stored, then more than one core is needed. Cores are arranged in matrix strung on a screen of wires, forming a core plane. Each core has an X and a Y wire running through it at right angles. It can be set to a 1 state by sending one-half the total required current through the X wire from left to right and the other half through the Y wire from top to bottom. The core can be set to a 0 state by sending the flow of current through the X and Y wires in the opposite directions. The polarity of the core in a specific direction, then, is determined by the direction and amount of current flowing through the X and Y wires.

Reading a Magnetic Core

2.8 When a particular core is read, one-half of the total amount of necessary amount is sent through each of the X and Y wires. In Fig. 4 one-half of the current is sent through the X-wire and the other half through the Y-wire. Only the core at the intersection of the two wires receives the full current. That particular core is said to be selected. All other cores strung on the X-wire are half-selected. They retain their status because each of them receives only one-half of the current necessary to magnetize them--that is, Y current only.

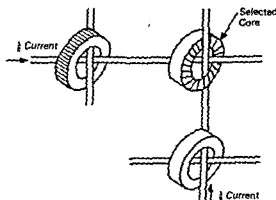


Fig. 4 Reading a Magnetic Core

Writing in a Magnetic Core

2.9 The process of writing data in a magnetic core is somewhat more

complex. Writing in a core involves either restoring the information previously stored in it or replacing the data with new information. When data is written in a core, one-half of the necessary current is sent through each of the X and Y wires in exactly the opposite direction from that used in a readout.

Thin-film Memory

2.10 Thin-film memory is produced by depositing very thin spots of metallic alloy on a ceramic or metal plate. This spot performs in the same manner as the core, except that only two wires are required. A typical thin-film element consists of a rectangle about 0.025 by 0.050 inches with a thickness of about 1,000 angstroms (0.0000004 in.). This rectangle can be thought of as a bar magnet. Applying a current which causes the polarity to rotate but not to flip induces a current which indicates whether a 1 or 0 was stored (Fig. 5). After the sense field is removed, the polarity is returned to its prior state by a digit pulse which "steers" the polarity back to a 1 or 0 state.

2.11 An overlay of etched copper wires provides the circuitry necessary to connect the individual elements with the circuits which read and write. The magnetic properties of film elements allow much faster switching times than can be achieved with cores.

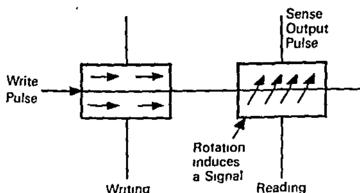


Fig. 5 Thin-film Memory

This provides faster memory cycle speeds, but the technical difficulties of thin films have, upto now, discouraged their use.

Plated-wire Memory

2.12 Plated-wire memory or woven-wire memory is a thin film which is

deposited around a fine wire. This wire carries the write current during a write operation. Data is retrieved in the sense line in a read operation. Insulated wires are woven across the plated wire in a fashion similar to weaving cloth. The area where the insulated wires loop around the plated wire forms a bit storage location (Fig 6).

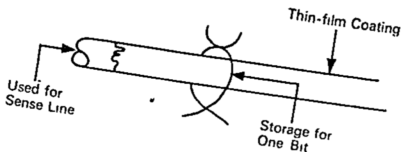


Fig. 6 Plated-wire Memory

Semiconductor Memories

2.13 Semiconductor memories consist of electronic devices etched on monolithic integrated circuit chips. A single silicon chip will contain memory circuits plus support circuits. The advantages are -- increased speed of memory access and reduced memory size (reduction of 50% or more). A semiconductor memory called "monolithic memory" by IBM, consists of silicon chips each of which contains 1434 circuit elements in an area less than one-eighth inch square. These elements form 128 memory circuits with support circuits. Semiconductor memories can be expected to increase in use and be a strong competitor for magnetic cores in main memory.

Virtual Memory

2.14 Each programme instruction and each data item to be used by the programmer--must be in the main memory during the time the instruction is executed and the data is used. The programmer usually refers to locations containing instructions or data by using a name or a label. The identification of the programmer instruction or data name with a specific memory location is performed by computer software. If a programmer writes a programme which is too large to reside in the main memory, the programme is segmented so that only part of it is in main memory at a time. The

remainder is put into secondary storage (such as magnetic disc) One of the trends in computer organization is to provide hardware and software which automatically perform segmentation and the related task of bringing programme and data segments from secondary storage into main memory when needed. This allows the programmer to ignore main memory limitations and act as if he had unlimited or virtual memory.

3. Computer Software

What is software ?

3.1 Software, in a general sense, means computer programmes, be they operating systems, languages, utilities, or user applications. Looked at another way, it is the tool with which applications are developed and implemented. These tools may be provided by the vendor supplying the equipment or by an independent vendor, or developed in-house.

3.2 For the purposes of our discussion, we will confine ourselves to a general study of various programming languages and their characteristics and some operating systems.

Programming Languages

3.3 At the outset, let us try and define a programming language. The ASA (now USASI) standard "Vocabulary for Information Processing (AA 66 b)" defines a *Programming language as* "A Language used to prepare computer programmes." Another definition (by the International Federation of Information Processing) states that it is "A general term for a defined set of *symbols* and rules or conventions governing the manner and sequence in which the symbols may be combined into a meaningful communication."

Machine-language and Symbolic Language

3.4 We found in the chapter on "Internal Operation of a Computer" that a computer basically operates on commands given in a series of 1s and 0s called machine-language. The biggest disadvantage of the machine-language was that the insertion or deletion of a single instruction (or piece of data) caused many--if not all-- of the addresses in other instructions to be incorrect. This situation could be improved somewhat by a scheme of *Relative addressing* or *Regional addressing*, in which the programme is divided into sections, each of which starts in a fixed location. Addresses within each section are given relative to the starting location. The desirability of using a symbolic code forced the development of symbolic assembly languages. These languages were also called *low-level languages.*

Some Drawbacks of Low-level Languages

3.5 The advent of symbolic languages was not enough to meet the demands of programmers. For one thing, programmers wanted to use other people's code wherever it was appropriate. This could not always be done because of differences in notation and lack of an effective way to link the pieces together. One of the main motivations for using other people's code was that certain programmes were being written over and over again. For example, square roots and trigonometric routines were being written by the dozens. In some cases, this proliferation was justifiable because one person was interested in saving space and therefore wrote as short a sub-routine as he could, while somebody else wanted to save as much time as possible. He, therefore, removed loops even at the cost of using more storage locations.

3.6 Programmers not only wanted the ability to use other people's code, but wanted the ability of easily bringing together small sections of a programme also. There was also an increasing demand for being able to write shorthands of various kinds.

Development of High-level Languages

3.7 All the above-mentioned drawbacks were tackled by various people in different ways. The first serious organised effort, however, started with the meeting sponsored by the office of US Naval Research in May 1954, which was called meeting on *Automatic Programming* Subsequently, various languages whose characteristics are described below were developed.

Characteristics of High-level Languages

3.8 In order that the high-level language may satisfy our requirements, certain characteristics need to be fulfilled. These are--

(a) *Machine Code knowledge not necessary :* A programming language should not demand knowledge of machine code. Also, the user need not learn about the registers available in the computer, nor the specific hardware instructions that are required to operate the [illegible]

run.

(b) *Potential for conversion to other computers :* A high-level language should possess some machine independence. In essence, there must be a reasonable potential of running a source programme written in that language on two or more computers.

(c) *Instruction explosion* : When a source programme written in a programming language is translated to the actual machine code [illegible]

(d) *Problem- oriented notation* · A programming language must have a notation which is somewhat closer to the specific problem being solved than in normal machine code. It should permit a relatively free format.To fall within the spirit of the concept of problem-oriented notation, a programming language must not require each statement type or executable unit to be specifically identified or flagged in a standard terminology or location.

Some Important High-level Languages

3.9, Higher level languages can be divided into two types--procedure-oriented and problem-oriented, although the distinction is not always maintained and both types are often referred to as "problem-oriented languages" or by the abbreviation POL. When used precisely, procedure-oriented refers to a language with which the programmer describes the set of procedures by which the problem is solved. With the problem-oriented language, the programmer uses the language to describe the problem itself, and the compiler translates this description into suitable computation procedures.

3.10 A POL language is designed for the programmer. Some of them are suited for relatively untrained persons so that they could programme their own problems: others are primarily for professional programmers. In both cases, the programmer writes programmes with little or no awareness of the specific equipment on which the problem will be run. The languages, so far as the programmer is concerned, should be machine-independent. This is not completely true because what the programmer wishes to do may be dependent on the availability of storage, I/O units etc., but in general, the concept is for machine-independent programming. The same POL programme may, therefore, be compiled and run on different computer systems with little or no change in the programme. A separate compiler, written especially for each of the computers is required. Compilers for the major languages are usually furnished by the manufacturer as part of the software support for the computer.

3.11 Some of the common high-level languages (though there are a number of high-level languages in use today) are--

(a) FORTRAN

(C) COBOL
(b) BASIC
(d) PL/1

Fortran

3.12 FORTRAN (an acronym for FORmula TRANSlator), is the most widely used procedure-oriented language. Almost every computer with memory size sufficient to implement FORTRAN has done so. The orientation of FORTRAN is towards problems which can be expressed in terms of arithmetic procedures or mathematical formulae. It works well for research and analytical problems in science, mathematics, and business. It is simple enough to be used by the non-professional programmer to programme his own problem solutions. FORTRAN was developed by IBM in 1954 for use in IBM 704 machines.

Basic

3.13 BASIC (Beginner's All-purpose Symbolic Instruction Code) is a language developed at Dartmouth College in 1965. This language was designed with a view to providing a stepping-stone for students to learn one of the more powerful languages such as FORTRAN. It is easy to learn and is especially useful in time-sharing. It is limited in capabilities and can be used with ease by a non-professional, occasional programmer.

A sample BASIC program for computing the value of simple interest, given the Principal, Rate of interest and Time Period

```
10    REM P=PRINCIPAL, R=RATE AND T=TIME PERIOD
20    INPUT P,R,T
30    LET I=P*R*T/100
40    PRINT "PRINCIPAL=";P; "RATE= ";R;
50    PRINT "TIME PERIOD= ";T; "INTEREST= ";I
60    END
```

These are line numbers. In some machines, these numbers appear on the screen of the Video Display Unit when the BASIC interpreter is called by the Programmer. In some other machines, a command called AUTO fulfils the same function.

Cobol

3.14 COBOL (the Common Business-Oriented Language) is a language providing a standard method for expressing solutions to business data

Note:- The reader may kindly refer to a book on BASIC or the Manual on BASIC supplied by computer vendors for further details.

processing problems. A basic characteristic of these problems is the existence of master files which are updated continuously or periodically. Business-type problems also tend to involve large volumes of input and output. COBOL was designed as an offshoot of a meeting convened in 1959 by the PENTAGON wherein manufacturers, users and universities were represented. The organisation of the COBOL programme is logical and, therefore, the language imposes some discipline on the programmer.

PL/1

3.15 PL/1 (Programming Language Version 1) is designed as a general-purpose language, combining the major features of both COBOL and FORTRAN and introducing features not found in either. The language has facilities for the real-time programmer in addition to the traditional commercial and scientific programming features. One of the difficulties encountered in a sophisticated language is the fact that many users do not need the advanced features and do not find it useful to be trained in their use. PL/1 handles this problem through a modular design and a default interpretation of specifications. Modularity means that the language is divided into modules or subsets for different applications and different levels of complexity. A programmer using one subset need not even know about the other modules.

APL

3.16 APL (standing for A Programming Language) was originally developed by K.E.Iverson in 1962 but was found to be very complex in its original form. APL/360, a modified version of the original APL is implemented primarily for time-sharing. Its users are enthusiastic and this may indicate a growing use. The language is oriented to formula type problems and therefore competes with FORTRAN.

4. How is a Programme Compiled ?

4.1 Though some elementary ideas on the development of various languages have been stated, any problem to be solved in a computer would normally undergo the following stages:

(a) *System design:* Design the computer system spelling out who would be the source and who the user based on the famous "How come, where gone" principle of the accountants. The format and frequency of both the input and output would also be decided during this phase of activity.

(b) *Planning the programme* : During the system design stage,

the various jobs would be broken down into programmes, and, description of the computer procedures required to perform the specified processing would be prepared (A programme, incidentlly, is a sequence of instructions telling the computer what to do). A programme flow chart is usually the most convenient method of preparing this description.

(c) *Coding* : The programme preparation activity continues with the coding of the logic described in the flowchart into procedure-oriented statements in the language being used. These are usually written on coding paper designed for the particular language. Such a programme is called the source-programme. These source-language statements are keypunched into punched cards or lines on the VDU screen, one line of coding into a card or one on screen. The resulting deck of cards forms the source-programme deck Of late, punched cards seem to be disappearing from the computer scene.

(d) *Translation (assembly or compilation)*· The translation phase begins with the loading of the translator routine of the compiler into the computer memory. The routine reads the source programme lines, interprets their meaning, and produces an object programme on some machine-readable output medium, such as punched cards, magnetic tape or magnetic disc. A listing of the programme is printed on the line printer or typewriter.

This listing includes any error diagnostics which the translator routine detects. The errors which can be detected are mainly textual errors involving misuse of the language. If errors are found, these must be corrected and the translation process repeated.

(e) *Debugging* : Debugging is the activity involving analysis of the errors in programme, both of the synctactical and the logic variety. The programmer traces through the logic in this phase to see if the programme does what it is intended to do.

(f) *Documentation* : This is one activity which is the most neglected one. Documentation would involve writing down, in a logical fashion, the essential features of the system. In this, higher level-languages are found to be better than lower-level ones.

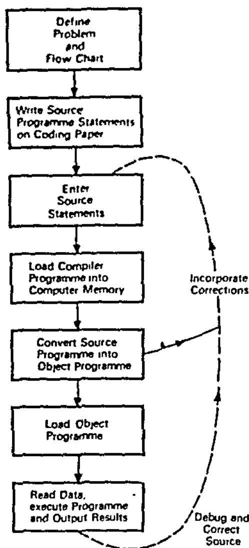


Fig. 7 Stages in Programme Development

4.2 The entire activity outlined above is pictorially represented in Fig. 7. While the user could write the programme and test it, we need an operator to operate the computer and run the programme. This is one area where a number of developments have taken place and let us see their impact on the management of computer installations.

5. Operating System

5.1 Since the advent of the computer era, users have had to share a computer facility, with the result queues used to be formed for doing various jobs. When users were permitted access to the computer, only one user could hope to tackle one problem at a time. It was also found that the user would keep the facility engaged even when certain corrections were to be made in his job processing. With the advent of multi-programming, where many users could share the computer concurrently, it became impossible for the operator to keep track of the jobs, schedule them, account for them, service them, etc.

5.2 Thus, operating systems came into existence to overcome difficulties of routine chores of the operators. Some of these are--

- (a) Programme loading.
- (b) Interrupt handling.
- (c) Scheduling of input/output operations.
- (d) Operator communications.
- (e) Job-to-job communications.
- (f) Job accounting.
- (g) Job scheduling and management.

What is an Operating System ?

5 3 An operating system, also called a "supervisor", "monitor" or "executive", is a set of routines written to manage the running of the computer. Operating systems range in complexity from simple systems which manage only basic functions to very complex ones. In general, the more sophisticated the computer system, the more complex the operating system required to manage its use. The philosophy underlying the operating system is that the computer should perform those operator tasks which it can do faster and more accurately and that the computer should be kept operating as continuously and effectively as possible.

Questions

1 What are the main types of storage ?
2. What are the characteristics of primary storage?
3. What are the types of primary storage ? Describe them.
4. Define a "Programming Language."
5. What are low-level languages and what are their drawbacks ?
6. What are the characteristics of high-level languages?
7. Name some programming languages.
8. How is a programme compiled ? Explain with the help of a diagram.
9. What is an operating system ?

5

Planning Tools for Computer Programming

1. Introduction

1.1 Before we undertake Programme-writing for a computer, it is necessary for us to define the problem. Computers, as described in Chapter 1, are nothing but systems which would do only those jobs which are clearly spelt out. Accordingly, one finds that computers have heralded an era of rational and precise thinking. No longer can the manager pride himself in his ability to write copious notes out-classing his colleagues in production of paper. The computer age demands brevity and clarity in problem definition.

1.2 Since the computers have started assuming the role of decision-making tools, it is necessary for us to see as to how best we could exploit them. Computer specialists have adopted the policy of first breaking down any problem into a sequence of steps, and then coding such steps in a language which the computer can understand.

1.3 While the specialists are comfortable with pictorial representation of decision-making procedures, the Executive seems to be more comfortable with a tabular form of representation.

1.4 In this section, it is proposed to discuss the various tools, available to a computer specialist and see if some of them could be used to assist the Executive decision-maker meaningfully.

1.5 It is proposed to analyse the tools under the following headings :

(a) Definition of a problem as a sequence of steps --use of Algorithms.

(b) Flowcharting techniques.

(c) Decision-Tables.

2. Definition of a Problem–Use of Algorithms

2.1 Before we embark upon a discussion of the various tools, it would be appropriate to consider as to how best a problem could be defined. One of

the toughest tasks confronting management in organizations since time immemorial is how to identify and define a problem. Once the problem is identified in its proper perspective, the various solutions to achieve the goal (or aim) could be worked out. When various alternatives present themselves before an executive, he has to make a choice. And herein lies his ability to isolate wheat from chaff. It is easier to define a mathematical problem than a social science one. Having defined the problem, one tries to solve the same using some procedure. Since the subject of decision-making using quantitative techniques is beyond the scope of this discussion, let us see in the first instance as to how we can evolve a few steps in solving a problem. For ease of presentation and comprehension, let us consider a simple mathematical problem.

Sieve of Eratosthenes

2.2 In this context, it would be better to seek recourse to the solution methodology propounded by Eratosthenes, a Greek mathematician in 250 B.C. for extracting prime numbers. A prime number, incidentally, is a positive integer, other than 1, that is exactly divisible only by itself and 1. Suppose we are asked a problem "What are all the prime numbers between 1 and some integer, say N". We would go about answering the problem by following a set of procedures. Such an unambiguous, complete procedure for solving a particular problem in a finite number of steps is called an Algorithm. (This word owes its origin to an Afghan mathematician by name Algoriwitz).

An Algorithm for the Sieve

2.3 The algorithm for solving the problem stated above would proceed as follows:

(a) Write down all the integers in numerical order from 2 [illegible] N and call this the Basic List.

(b) [illegible] us call it M, then cross it off the basic list and every Mth number thereafter.

(c) If M is less than $\sqrt{N}$, return to (b); if M is $\geq \sqrt{N}$, add all the remaining uncrossed-out numbers to the prime list and then stop.

(d) The prime list so prepared is in fact the desired list of primes less than N.

2.4 Let us see the validity of the above algorithm by taking an example. Suppose we wish to find out all the primes less than 30. We record numbers as shown *in Fig.1.*

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Fig. 1. First Step in the Solution of the Problem (Basic List)

2.5 Next, we record 2 in the Prime List and cross out every even number. Since $2 < \sqrt{30}$, we record 3 and cross out 3, 6, 9, 12, etc. Since $3 < \sqrt{30}$, we go back to the beginning of the basic list, place 5 on the prime list, and cross out every fifth number. Since $5 < \sqrt{30}$, we go back to the beginning of the Basic List and place 7 on the Prime List. Since $7 > \sqrt{30}$, we stop the procedure and write down all the uncrossed numbers also in the Prime List. We thus find that 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29 are the prime numbers less than 30.
(Please refer to Fig. 2 for the solution).

Basic List	*Prime List*
~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~	2, 3, 5, 7, 11, 13,
11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~	17, 19, 23, 29
~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~	

Fig. 2. Solution of the Problem Till $M > \sqrt{30}$.

2.6 From the above example, one finds that steps (b) and (c) were repeated a number of times while solving the problem. Such problems, where some steps are repetitive in nature are ideally suited for solution by computer. A human being would get tired while doing repetitive tasks and thus commit mistakes after a few repetitions, whereas the computer never complains of mental fatigue.

3. Technique of Iteration

3.1 It is, therefore, appropriate to construct an algorithm, wherever possible, by taking advantage of the repetitive steps if such steps could be easily identified. In the example just discussed, one could repeat step 3 and step 2 any number of times till the solution was reached. In some other applications, however, repetitive calculations are designed to obtain better and better guided estimates of a desired result. Successive estimates of the quantity are obtained by substituting previous estimates in the same procedure. Such a technique is called iteration. Very often, only the last estimate of the quantity is used to obtain the next estimate. The problem is to find a valid procedure, choose an appropriate starting value and decide on

to find a valid procedure, choose an appropriate starting value and decide on when to stop the iteration. Though one may choose a procedure and a starting value appropriate to the problem, stopping the iteration at a suitable point of repetition poses a problem One of the methods used is to assume that as the iteration gets nearer and nearer to the true value, successive estimates would differ by smaller and smaller amounts. One could prescribe a small quantity and stop the iteration when successive estimates differ by less than that quantity. We could also stop the procedure once we have done a certain specified number of iterations.

Iteration Procedure using Newton's Formula for Finding Square Root.

3.2 The above statements can best be illustrated by an example. Suppose we wish to find the square root of 51, we must decide on a procedure first. One of the methods for finding the square root is based on Newton's formula wherein the next estimate of the square root of a number X (say S_{n+1}) is obtained by using the formula

$$S_{n+1} = \frac{1}{2}\{S_n + X/S_n\}$$

where S_n is the previous estimate

In the problem before us, we are aware that 7X7 is 49, and 8X8 is 64 Hence, a starting value of 7 for Sn seems appropriate. The error we wish to permit between two successive values is say 0 0001. Then

$S_1 = \frac{1}{2}\ \{S_0+51/S_0\}$ Where $S_0=7$

Step 1

$S_1 = \frac{1}{2}\{7+51/7\}$

$= \frac{1}{2}\ (7+7.2857)=14\ 2857/2=7.1429$

Step 2

$S_2 = \frac{1}{2}\ \{S1+51/S1\}$ Where S1=7.1429

$= \frac{1}{2}\ \{7.1429+51/7.1429\}$

$= \frac{1}{2}\ (7.1429+7.1399)$

$= \frac{1}{2}\ X14.2828=7.1414$

Step 3

$S_3 = \frac{1}{2}\ \{7.1414+51/7.1414\}$

$= \frac{1}{2}\ (7.1414+7.1414)=7.1414$

In this example, we have been able to arrive at two consecutive estimates which are equal, the error being less than 0 0001. The reader should not get away with an impression that all problems are as simple as the one illustrated. In fact, the speed with which one arrives at a solution depends to a large extent upon the value one is able to assign to the initial estimate. Let us now see as to how the algorithm could be converted into a diagrammatic form.

4. Pictorial Representation–Flowcharts

4.1 When the algorithm has been chosen, it is convenient to represent the various steps in the form of a diagram, called Flowchart, before encoding the steps in a language which the computer would understand. Flowcharts are widely used because they combine the precision and continuity of an ordinary algorithm with the visual aid inherent in a diagram. What is more, they highlight the dead-ends, unforeseen alternatives, and special subsections of the algorithms that are susceptible to errors.

4.2 A flowchart, in fine, is a graphic representation of the operations, and decision logic, and the order in which they are to be handled. In effect, it is a detailed outline giving the sequential steps which must be performed by the computer for the accurate processing of business data. As a graphic representation, the flowchart enhances communication between one person and another, aids the computer specialist in visualising the sequence of the necessary operations, and acts as a guide for coding the symbolic or procedure-oriented language (which it contains) into machine-language. The flowchart also becomes a part of programme documentation when the programme is "debugged", thus serving as a primary source for future reference. A programme flowchart, thus, is a means of outlining a given problem.

Types of Flowcharts

4.3 Flowcharts can range from general ones to detailed ones showing the logic of the individual programme steps In the analysis which precedes the coding of programme steps, two types of flowcharts may be prepared (viz.) the system flowchart and the programme flowchart. The system flowchart shows the flow of data through all parts of a data processing system. The exphasis is on how data processing is to be accomplished (i.e.) how source media can be converted to output media. The programme flowchart, on the other hand, shows the operations and decisions to be followed by the computer in a particular programme. The emphasis is on the programme logic and processing sequence. The same set of flowchart symbols can be used for both types of flowcharts. Programme flowcharts can be further subdivided into macroflowcharts and microflowcharts.

Flowcharting Symbols

4 4 One could always make his own symbols for flowcharting, but this is neither necessary nor desirable. The computer industry has agreed on a standard set of symbols, and these should be used. These symbols have been in general use since 1967. A list of flowcharting symbols is shown in Fig. 3.

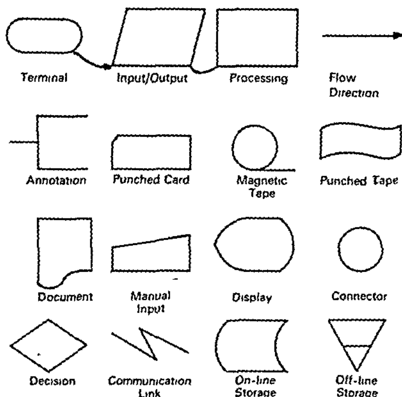


Fig. 3. Some Common Flowcharting Symbols

Let us illustrate the use of various symbols with the help of an example.

4.5 Suppose we wish to read a set of employee records of an organisation maintained on magnetic disk. Each record consists of details like employee number, date of birth, date of entry in the organisation, sex

(1=male and 2=female), marital status, number of children, basic pay etc. Let us suppose that the organisation wishes to give a nominal bonus of Rs. 10/- per month, as part of a family-planning programme to the employees who fulfil the following criteria:

(a) Must have served the organisation for more than 5 years.
(b) Must be older than 25 years
(c) Must be a male.
(d) Must have less than 3 children.
(e) Basic pay must not be more than Rs. 600/-

We are required to draw a detailed flowchart for a programme to compute bonus for eligible employees and print the total bonus required to be paid. The flowchart for this programme would run somewhat as shown in Fig. 4.

4 6 Some of the symbols which we have used in Fig.4 are :

(a) Terminal
(b) Input/Output symbol.
(c) Process symbol.
(d) Annotation symbol.
(e) Decision symbol.
(f) Connector symbol.

4.7 Let us see the meanings of the various symbols mentioned above i greater detail–

(a) *Terminal*: Normally denotes the beginning and end of a programme. Each flowchart s hould normally begin with the terminal symbol at the upper left corner of the flowchart sheet.
(b) *Input/Output symbol* Stands for an instruction to either an input or an output device. The input device transfers data to the computer for processing. The output device, on the other hand, prepares the result of the processed information to the user. In this example, reading a card is an input operation while printing the totals is an output operation.
(c) *Process symbol* denotes an operation involved in the actual processing of data It is used in manipulating data and in performing arithmetic opertions within the computer. In this example, addition, subtraction etc.,areprocess operations. Since these operations are carried out in the central processor, the process symbol is also called the "stored programme symbol.

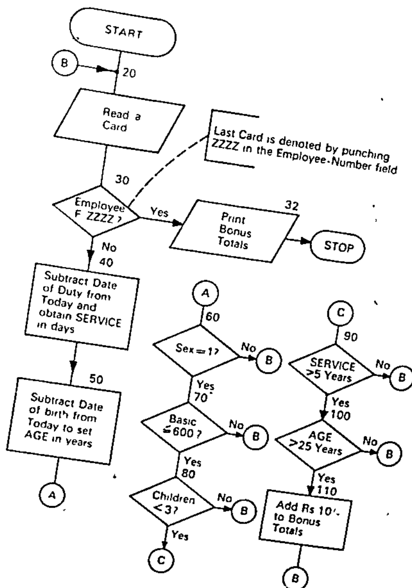


Fig. 4. Bonus Flowchart

(d) Annotation symbol is used to give descriptive comments or explanatory notes for clarification and are not meant for signifying any action in the programme.

(e) Decision symbol is usually located somewhere between the

input and output symbols, depending upon the type of

represents the "thinking" part of a computer and relates to algebraic notations used to test a given relationship.

(f) Connector symbol is a non-processing symbol used to connect one part of a flowchart to another without drawing flowlines. It denotes an entry from, or exit to, another part of the flowchart. When used, it conserves space by keeping related blocks near one another, reduces the number of flowlines in complex programmes, and eliminates cross-over from taking place.

Cross-referencing

4.8 After a flowchart is drawn, it is customary to give each block a number based on its sequence in the flowchart. This optional step, called cross-referencing, can be used later as a statement number in the coding phase of the programme preparation. The increment between any two blocks in the main programme should be constant and sufficient for labelling secondary instructions.

Hints to Flowcharting

4.9 Some useful hints found helpful in constructing a programme flowchart are as follows :

(a) The flowchart should flow from top to bottom and from left to right.

(b) The language used in describing each of the steps in the flowchart should be normal English, and not machine-language. Such an approach enables other people to understand the steps better.

(c) In complex programmes, connectors should be used to minimise the crossing of flowlines.

(d) Intersecting lines should be avoided as much as possible.

(e) The flowchart should be legible, and must use a standard template so that the diagram will look neater and more presentable than a hand-drawn flowchart.

Some Drawbacks

4.10 Flowcharting is time-consuming, and, in cases involving complex

programmes, calls for a number of revisions. These factors, and the tight schedule imposed on the programmers give rise to shoddy flowcharts. When the flowchart is messy, or, when it is altogether done away with, one finds it difficult to code or debug the programme. Another drawback is due to lack of a standard method of construction. During the preparation of a programme flowchart, a programmer often finds himself under a strain, with a long list of ideas and data, most of which are directly inter-related. All these factors contribute towards the flowchart not fulfilling its role. Flowcharting tends to become both problem and computer-oriented, particularly if, flowcharts are written after the programme has been run, which unfortunately is often the case. Documentation should, instead, reflect decision patterns–the why, rather than the how.

5. Decision Tables

5.1 While flowcharts are useful in helping the computer specialist to represent the problem pictorially, decision tables are graphic displays that fill the communication gap between the operating manager and the computer specialist. They also document the information needs of operating management in a form directly usable by the specialist. They may be used in place of, or along with, flowcharts and block diagrams. A decision table (sometimes referred to as *decision logic table*) is also an effective tabular approach used in (a) planning programmes (b) showing cause-effect relationships, and (c) handling situations that involve complex decision logic. It presents the original conditions and the course of action to be taken if the conditions are met.

Structure of a Decision Table

5.2 A decision table consists of four basic sections. A horizontal line divides the decision table into two major sections (viz.) conditions and actions The *conditions* may be thought of as *if* statements, while the actions may be thought of as *then* statements. A vertical line divides the table into two other major sections and the left side is called the *stub* while the right side is called the entry. The stub portion describes and names the conditions and actions in which we are interested and the entry portion specifies the logical relationships.

5.3 Horizontal levels, called *rows*, run across the entire table. To aid in identifying them, they may be assigned letters or numbers. The entry portion is sub-divided by vertical lines to form columns, called *rules*, which may also be assigned identifying numbers. On top of the condition stub is a special row called the *Table Header*. The last row, of the table is used for any remarks. Fig. 5 denotes the four basic sections of a Decision Table.

Table Header	
Stub	Entry
CONDITIONS	
ACTIONS	
Stub	Entry
Remarks	

Fig. 5 The Basic Sections of a Decision Table

Rules

5.4 The condition stub lists all conditions which are pertinent to a given problem and the condition entry documents all possible combinations of the specified set of conditions which we must consider. The action-stub lists all possible actions which might be taken, while the action entry specifies which action we must take. The particular action we take in response to a particular combination of conditions forms a *rule*. The conditions and actions are linked in the rules.

Types of Decision Tables

5.5 Decision tables are generally classified by the information recorded in the entries. They are basically of three types – limited-entry or mixed-entry.

Limited-entry Tables

5.6 These are very widely used and their approach [illegible]

blank. The condition stub must be written so that it is either true or false and the action stub must completely describe the action to be taken. Fig. 6 is a typical example of a limited-entry table.

5.7 The problem is to ascertain whether the request by a customer for loan could be granted or not. In this case, the statements in the condition stub are written in a question form requiring either a *yes* or *no* response. The response to each statement is written in the condition entry. Likewise, the action is written in a narrative form in the action stub, with "X" in the action entry responding to each "Y" in the condition entry. Being limited to a Yes or No alternative as is the case in binary logic, limited-entry tables are ideally suited for computer-oriented applications.

	Bank Loan Table	1	2	3	4
IF	Credit record excellent?	Y	N	N	N
	Monthly salary over Rs.1500/-?		Y	N	N
	Assets worth Rs.30000/-?			Y	N
THEN	Approve loan for 10000	X			
	Approve loan for 8000		X		
	Approve loan for 5000			X	
	Reject request				X

Fig. 6. Limited-entry Decision Table

Extended-entry Tables

5.8 In this case, the statements written in the stub section are extended into the entry section. The condition stub identifies the elements to be tested, while the condition entry describes those elements or defines the value in absolute or relative form. Similarly, the action stub states the action. Fig. 7 depicts an extended-entry table for the one provided in Fig. 6.

Bank Loan Table

Credit Record	Excellent	Good	Fair	Poor
Monthly income		≥ Rs.1500/-	< Rs.1500/-	<Rs.1500/-
Assets			≥ Rs. 30000/-	< Rs. 30000/-
DECISION				
P. Approve loan for 10000	P	Q	R	S
Q. Approve loan for 8000				
R. Approve loan for 5000				
S. Reject request				

Fig. 7. Extended-entry Decision Table

This type offers the advantage of reducing the number of items listed in the condition and action stubs, while essentially providing the same information. Extended-entry tables resemble conventional data tables and are ideally suited to problems that have few variables, with each variable having many values.

Mixed-entry Tables

5.9 As its name implies, a mixed-entry table is a combination of a few rows with extended entries and a few other rows with limited entries. While the rows may be mixed, each row in itself must be either entirely extended or entirely limited. In Fig. 8, while rows 2 and 3 have an extended-entry format, the remaining rows have a limited-entry format. Conversion from either extended – entry or mixed–entry is done by writing the value of each element as an independent condition or action and assigning 'Y's, 'N's or 'X's wherever applicable.

Bank Loan Table	1	2	3	4
Credit Record Excellent ?	Y	N	N	N
Monthly income ?		≥1500	<1500	<1500
Assets ?			≥30000	<30000
Approve loan for 10000	X			
Approve loan for 8000		X		
Approve loan for 5000			X	
Reject request				X

Fig. 8 Mixed-entry Tables

A Comparison

5.10 In the Limited-entry table, the status of each condition is either true or false, and each action is either applied or not applied, thus making it more precise than either the extended-entry or the mixed-entry table. This table also facilitates logical analysis and development of clerical or computer procedures. The extended-entry table is best suited for problem solving, and offers the possibility of having two or more responses to a given condition.

Advantages of Decision Rules

5.11 Decision tables are of some use to everyone - Programmers, Analysts, Supervisors and Managers. As a documentation tool, they provide

a more concise and simpler form of data analysis than either the narrative or the flowchart form They can be effective tools for extracting details pertinent to a given problem and then for formulating questions. Unlike flowcharts which are developed around the programmer's logical thinking, and, are arbitrarily made simple or complex, decision tables bind the person preparing them to be concise and, to consider all sorts of conditions. When completed, they serve as an easy-to-follow communication device between technical and non-technical personnel. They are verbally-oriented for managers, and logically prepared for programmers and analysts. Furthermore, they are easy to learn and update They continue to function, once the logic is developed

Decision Tables-to-computers

5.12 Decision Tables may be conveniently coded into computer programmes Hence, many computer manufacturers provide some software to convert standard decision tables to programmes. Notable among these are DETAB/65, DETRAN (DECISION TRANSLATOR) and so on, which help the user to state the problem in a pre-determined format. Further, after incorporation in a high-level language and compilation, they provide a meaningful programme. Thus, the mundane clerical chores in programme writing can be avoided.

Questions

1. What is an algorithm ?
2. Describe an algorithm for the "Sieve of Eratosthenes."
3. What is iteration ?
4. Using Newton's formula, find the square root of–
   - (a) 49
   - (b) 63
   - (c) 71
5. What are flowcharts ? What is the flowchart symbol for –
   - (a) Decision Stage.
   - (b) Printing Stage.
   - (c) Communication Link.
6. What are the flowcharting conventions ?
7. What is a Decision Table ? How does it compare with a Flowchart ?
8. What is the structure of a Decision Table ?
9. What are the types of decision table ? How do the different types compare with each other ?

GLOSSARY AND THE EMERGING SCENE

Since a few terms have been brought into common usage in the recent past, a limited glossary of a few of these terms is given below. Incidentally, some terms have already been defined in the earlier chapters.

A small note on the emerging scene is also enclosed.

"On-line"

When electronic computers were designed initially, most of the data received from places outside the computer room had to be brought physically. In other words, the development of data communications had not reached any meaningful stage during the era of the First and Second generation computers. Thus, most of the devices (such as data preparation devices) were "off-line".

With the advent of reliable data communications, data could be transferred from one place to another by means of telephone and telegraph channels. Thus, many of the devices (particularly terminals) could be located far away from a computer and connected to it for interaction "On-line". Air-line and railway reservations are "On-line".

"On-line" refers to physical connection only. One can say with a reasonable degree of confidence that the "On-line" connections blossomed with the advent of 3rd generation of computers.

"Real-time"

While "On-line" refers to physical connection alone, "real-time" refers to the facility provided which enables a user or a machine to access the computer, verify whether the operation is proceeding smoothly against pre-determined standards and then make corrections (if needed) in as short a time as possible. For example, let us imagine the status of a space vehicle. The path to be followed by the space vehicle is stored in it. The actual path traced by the space vehicle would be transmitted to the computer of the ground control station periodically. The computer at the ground control station will then compare the pre-determined path with the actual path and give corrections to the space vehicle in a split second as it were. This process is called "real-time".

"On-line" is a term normally meant to distinguish "off-line" for connection, update and query retrieval. "Real-time" on the other hand signifies control. While "real-time" corrections can be effected only if the concerned device is "on-line", "on-line" queries at the airlines reservation counter or the railways reservation counter need not be "real-time".

"Maxi-Computers","Midi-Computers" and "Mini-Computers"

When the electronic computers were originally designed in the Forties, the then designers could not visualize the exciting growth potentials of miniature devices. Either because of this fact or due to others, languages used with computers also remained at the primitive level. One used machine language, symbolic language etc. In addition, the number of bits (binary digits) which were transferred from one device to another (or from one device to the Central Processor or vice versa) happened to be in groups of 4, 8 etc.

The concept of "words" emerged with the machines made by the Digital Equipment Corporation (DEC), USA; International Computers Limited (ICL); UK; Honeywell Corporation, USA etc. While DEC machines used a group of 36 bits, ICL used a group of 24 bits and Honeywell used a group of 48 bits for transferring them from one device to another.

With the advent of semi-conductor memories and better circuit design, machines with 8-bit groups for transfer of data and instructions also emerged. At about the same time (late Sixties and early Seventies) 16-bit machines also came on the scene.

One did not know how to distinguish the various types. Herein came the model of skirts worn by girls t the rescue of computer designers and manufacturers. Just as "maxi", "midi" and "mini" skirts indicate the length of skirts worn by girls, the same terms were used to indicate computers with different word lengths. TDC-316 made by the Electronics Corporation of India Limited (ECIL), Hyderabad could be considered to be a midi.

Unfortunately, the sanctity of these terms has been eroded due to the advent of micro-processors, which now are able to claim a large word length. In addition to the length of the word dictating the nomenclature of a computer, it (the length) also determines the largest number (mainly of the address of a location) that the word can accommodate.

An 8-bit machine can store an address upto a number 255 ($= 2^8 - 1$) while a 16-bit machine can store a number upto 65375 ($= 2^{16} - 1$).

"Maxi" computers can also accommodate larger number of peripheral devices "on-line" at the same time as compared to "Mini" computers.

The cost of "Maxi" computers is higher than the "Mini" computers for a number of reasons explained above.

"Super" Computers

"Super" is a title given to very large computers which have the following characteristics:-

(a) A large internal memory–say–12 Mega Bytes (a byte is a group of 8 bits) and above

(b) A very fast cycle time, switching time etc. In fact, the speed is indicated in the form of Million Instructions Per Second (MIPS) — say in the range of 500 MIPS and above. The other form of speed representation is Millions of Floating Point operations Per Second say (FLOPS). 100 Mega FLOPS and higher are some indications for qualifying as "Super" computers.

(c) An ability to handle a large number of peripheral devices and terminals without any consequent reduction in speed of access of the "Super".

(d) Front - End Processors (FEPs) to handle telecommunication (Data communication) chores with distant/remote terminals.

(e) Back-End Processors (BEPs) to handle access to, and retrieval of, data stored in secondary storage devices as magnetic disks, drums etc.

Micro-processors

Micro-processors are also computers but they mainly make use of semi-conductor memories. The advantage of a semi-conductor is that the speed of switching from one stable state (say 1) to another (say 0) is much higher than that found with magnetic core memories.

Personal Computers (PCs)

Sometime in the late Seventies and early Eighties, small manufacturers in U.S.A. introduced the concept of Personal Computers for helping users to store data relevant to them. Data such as train timings, Income-tax details, property details, simple tabulations, standard letters, small research problems and solutions, standard formulae, conversion details such as currency, units of measure etc. were needed to be made available on tap, as it were, to various people.

The emergence of cheap ICs (Intergrated Chips) and Large Scale Integration (LSI) Chips proved to be a boon for the growth of P.C.s.

In addition to the increasing economy and sophistication of electronic circuitry, one also found that one could connect the computing device to the standard home TV screen and use it as a video display unit. All one needed was a standard cassette tape recorder and a printer (if a print-out was needed).

Manufacturers like APPLE, COMMODORE, SINCLAIR etc. entered the PC market and thus the PC became very easily and cheaply available by the mid-eighties.

The International Business Machines (IBM) wrested the initiative of small manufactures and introduced its IBM-PC. The IBM-PC and its other improved versions such as IBM-PC/XT etc. have become the industry standard today, what with a large number of software programmes made available worldwide by various IBM user groups. These software programmes and languages have forced most of the other manufacturers of PCs to claim that their machine is IBM-PC compatible or IBM-PC/XT compatible.

Where are we now ?

Today, one is not able to very clearly demarcate the line between a "maxi", "midi" or "mini" with as much confidence as one would like to possess. In fact, many micro-processors are 16-bit or 32-bit machines.

May be the only direction in which one is heading , is towards the Fifth Generation which would, hopefully, emerge at least with the dawn of the 1990s. These Fifth Generation types may not be computers in the strictest sense of the term but may be "Intelligent machines" trying to translate sentences from one language to another, recognize human voice, read handwritten statements etc.

One thing is certain, however. In this era, one should not ask if he should buy a computer (they are available in plenty in various sizes and shapes and to suit almost every pocket) but should ask himself as to what he would do with the computer !

Addresses of some suppliers of Microprocessors and PCs.

1. Blue Star Ltd.
   Computer Systems Deptt.
   Bhandhari House
   91 Nehru Place
   New Delhi-110 019

2. CMC Ltd.
   Jeevan Vihar
   3 Sansad Marg
   New Delhi-110 001

3. Chowgule Industries
   Chowgule House
   Mormagao Harbour
   Goa-403 803

4. DCM Data Products
   2nd Floor
   8E Rani Jhansi Road
   New Delhi-110 055

5. Electronic Corporation India Ltd.
   Computer Division
   2nd Floor, Gopala Tower
   25 Rajindra Palace
   New Delhi-110 008

6. Eiko Sales Pvt. Ltd.
   102/1 Residency Road
   Bangalore-560 025

7. ESPL,
   A1/152 Safdarjung Enclave,
   New Delhi- 110029

8. Hinditron Computer Pvt. Ltd.
   204-206 Hemkunt Tower
   98 Nehru Place, New Delhi-110 019

9 Hindustan Computer Ltd.
RM 512, A97 Nehru Place
New Delhi-110 019

10 International Data Management
4 Nehru House,
Bahadur Shah Zaffar Marg,
New Delhi-110 002

11 International Computer Indian Manufactures Ltd.
Jeevan Tara Building
5 Sansad Marg
P.O Box 76
New Delhi-110 001

12. Indchem Electronics Ltd.
Data Devices Division
Sudarshan Gardens
102 Velachery Road, Guindy
Madras-600 032

13. Keltron
2nd Floor, Hemkunt Tower
98 Nehru Place
New Delhi-110 019

14. Minicom Pvt Ltd.
D-11 Everest, Tardeo
Bombay-400 034

15. NELCO, 5th Floor
Deen Dayal Upadhyay Building
7E Rani Jhansi Road
New Delhi-110 055

16. Operations Research Group
33 Basant Lok
Vasant Vihar
New Delhi-110 057

17. OMC Computer Ltd., 5th Floor,
Chenoy Trade Centre,116 Park Lane
Secunderabad, 500 003 Andhra Pradesh.

18. PAC Systems
    914 Chiranjiv Tower
    43 Nehru Place
    New Delhi-110 019

19. PSI Data Systems Ltd.
    C-11 SDA Shopping Centre
    Opp. IIT Main Gate
    New Delhi-110 016

20. Tata Burroughs Ltd.
    Manish Commercial Centre
    216A, Dr. A.B. Road
    Worli, Bombay-400 025

21. Uptron India Ltd.
    D-6 South Extension - II
    New Delhi-110 049

22. USHA Microprocessors
    105 Chiranjiv Tower
    43 Nehru Place
    New Delhi-110 019

23. Wipro Information Technology Ltd.
    55 Basant Lok
    Vasant Vihar
    New Delhi-110 057

24. Zenith Computer Ltd.
    Zenith House
    35 Pusa Road
    New Delhi-110 005